PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : C12Q 1/68, G06F 17/17 | A1

(11) International Publication Number: WO 98/11258

(43) International Publication Date: 19 March 1998 (19.03.98)

(21) International Application Number: PCT/US97/16933

(22) International Filing Date: 16 September 1997 (16.09.97)

(30) Priority Data:
60/025,241 16 September 1996 (16.09.96) US

(71) Applicant: UNIVERSITY OF UTAH RESEARCH FOUNDATION [US/US]; Suite 170, 416 Wakara Way, Salt Lake City, UT 84108 (US).

(71)(72) Applicant and Inventor: MARKS, Andy, F. [US/US]; 175 South West Temple #530, Salt Lake City, UT 84101 (US).

(74) Agents: CHRISTIANSEN, Jon, C. et al.; Van Cott, Bagley, Cornwall & McCarthy, Suite 1600, 50 South Main Street, Salt Lake City, UT 84144 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR ANALYSIS OF CHROMATOGRAPHIC MIGRATION PATTERNS

(57) Abstract

The present invention includes a method and apparatus for the detection and analysis of information-containing signals in chromatographic data using iterative blind deconvolution and fuzzy logic algorithms. The invented method analyzes chromatographic data from a wide variety of sources of DNA sequencing information, including gel and capillary electrophoresis. Autoradiograms, single-fluor, four-lane and four fluor, single lane fluorescent chromatographic data are suitable sources of unprocessed input data. The output from the invented base calling method includes called (identified) sequence data and quality values for the called bases.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHOD AND APPARATUS FOR ANALYSIS OF CHROMATOGRAPHIC MIGRATION PATTERNS

I. Background of the Invention

A. Field of the Invention.

This invention relates to the field of signal detection and analysis of chromatographic migration patterns as commonly applied to mixtures of molecules. More specifically, this invention relates to a method and apparatus for signal detection and analysis of chromatographic migration patterns as applied to the determination of DNA sequences.

B. Description of Related Art.

The ability to efficiently and accurately detect and analyze information-containing signals in chromatographic data is important for handling large amounts of data. Such an ability is particularly important for projects such as the Human Genome Project, where large amounts of information will be generated which must be analyzed and integrated to produce a representative sequence of an entire human genome. To expedite the analysis of DNA sequence information, numerous methods have been developed. For example, a U.S. patent to Clark Tibbetts (No. 5,365,455) discloses a method for the automated processing of DNA sequence data. This patent is incorporated by reference herein in its entirety. The Tibbetts' method derives information from informative variables obtained from the input data set. Such informative variables may include the relative intensities between adjacent signals, the relative signal spacing and pattern recognition factors.

The Tibbetts' method is limited, however, by the quality of the chromatographic data. Tibbetts' method relies to a certain extent on the reproducibility of chromatographic data to train the base identification ("calling") system. The apparatus generating the chromatographic data, therefore, needs to be consistent from run to run to avoid retraining the algorithm. Because chromatographic data frequently contain background noise and migration aberrations which obscure information-containing signals, analyses based on signal spacing may produce errors in signal identification. Similarly, because signal intensity often varies in an unpredictable manner, signal identification based on intensity may also result in significant identification errors.

A U.S. patent of Thomas Stockham and Jeff Ives (No. 5,273,632) discloses an alternate method for base identification using blind deconvolution ("BD"). This patent is incorporated by reference herein in its entirety. The method of Stockham and Ives uses blind deconvolution to deblur information-containing signals in chromatographic data. This method, however, is significantly limited in the following manner. First, it relies on data derived from scanned autoradiogram image data. Second, the method requires user input of the BD filter bandwidth and programmer alterations to various thresholds. Third, the Stockham and Ives method does not adequately deal with lane to lane mobility differences. Fourth, the insertion/deletion and correction logic was too simple. Fifth, the putative peak detection was based on thresholds, and therefore, could miss band detections when band amplitudes dropped below the threshold. Sixth, the method of Stockham and Ives lacked the ability to align and merge adjacent sample segments. Finally, that method lacked band quality measures useful in automatic data routing and or sequence assembly.

II. Summary and Objects of the Invention

The present invention includes a method and apparatus for the detection and analysis of information-containing signals in chromatographic data. The invention also includes a method and apparatus for detecting and sharpening signal peaks in chromatographic data. It is an advantage of the present invention that a chromatographic data from a wide variety of separation processes can be analyzed. Such separation processes include, but are not limited to, gel and capillary electrophoresis.

The present invention includes the steps of preprocessing signal data, reading successive sample segments, selecting blocks of high quality sequence and then producing traces of aligned high quality sequences. It is an advantage of the present invention that the chromatographic data may include single fluor samples fractionated in multiple lanes and multiple fluor samples fractionated in single lanes.

It is an object of the present invention to provide a method for preprocessing chromatographic data by baseline subtracting background noise. It is an advantage of the present invention that the method of baseline subtraction may be varied according to the type

of chromatographic data being analyzed. It is a further advantage of the invention that sparse chromatographic data may be interpolated during preprocessing.

It is an object of the present invention to read the preprocessed signals in successive sample segments. It is an advantage that the sample segment size may be sufficiently large to provide for rapid and efficient signal analysis.

It is an object of the invention to provide a method and apparatus for detecting information-containing signals which are not uniformly distributed in the chromatographic data. This analytic technique uses iterative blind deconvolution to determine band frequency in sample segments. It is an advantage of the invention that the filter-band width is automatically varied during iteration to optimally detect the signals in the preprocessed chromatographic data. It is a further function of the invention to detect and correct signal data derived from chromatographic data which have segments which are short in one or more signal types (for example, "band-lite" signals).

It is an object of the present invention to provide a method and apparatus to detect and correct for mobility differences. It is a feature of the invention that mobility differences are corrected using a Monte Carlo alignment rather than using band position or spacing information. It is an advantage of the present invention that the Monte Carlo alignment is an iterative process to optimize signal alignment.

It is an object of the invention to enhance band detection using fuzzy logic. It is a feature of the invention that band detection is performed using fuzzy logic blocks, each block providing a particular method of data analysis. It is an object of the invention that each fuzzy logic block may be optimized for a particular analytic function.

It is an object of the present invention that the invention may optionally provide a quality measure for each signal. It is a feature of the invention that the quality measure can be utilized during subsequent alignment steps. It is an advantage of the invention that the quality measure can provide left and right cutoff point to limit subsequent analysis to data above a given quality measure.

These and other objects, features and advantages of the invention will be clear to a person of ordinary skill in the art upon reading this specification in light of the appending drawings.

III. Brief Description of the Drawings

Figure 1 depicts a flow chart for the invented base calling method.

Figure 2 depicts a flow chart of the preprocessing step of Figure 1.

Figure 3 depicts a flow chart of the base reading step of Figure 1.

Figure 4 depicts a flow chart of the extra-normalization step of Figure 3.

Figure 5 depicts a flow chart of the peak detection and refinement step of Figure 3.

Figure 6 depicts a flow chart of the OmitOkN fuzzy logic block of Figure 5.

Figure 7 depicts a flow chart of the OKSpMembership fuzzy logic block of Figure 6.

Figure 8 depicts a flow chart of the OmitOkN Bad Spacing Membership fuzzy logic block of Figure 6.

Figure 9 depicts a flow chart of the OmitOkN Cross Banding fuzzy logic block of Figure 6.

Figure 10 depicts a flow chart of the OmitOkN Height fuzzy logic block of Figure 6.

Figure 11 depicts a flow chart of the GapCheck fuzzy logic block of Figure 5.

Figure 12 depicts a flow chart of the GapCheck Gap Membership fuzzy logic block of Figure 11.

Figure 13 depicts a flow chart of the GapCheck Width Membership fuzzy logic block of Figure 11.

Figure 14 depicts a flow chart of the Monte Carlo Alignment function of Figure 4,

Figure 15 depicts a flow chart of the BaseQual fuzzy logic block of Figure 1.

Figure 16 depicts a flow chart of the BaseQual Height Membership fuzzy logic block of Figure 15.

Figure 17 depicts a flow chart of the BaseQual Cross Banding Membership fuzzy logic block of Figure 15.

Figure 18 depicts a flow chart of the BaseQual Width Membership fuzzy logic block of Figure 15.

Figure 19 depicts a flow chart of the BaseQual Shape Membership fuzzy logic block of Figure 15.

Figure 20 depicts a flow chart of the BaseQual Baseline Buzz Membership fuzzy logic block of Figure 15.

Figure 21 depicts a flow chart of the BaseQual OK Spacing Membership fuzzy logic block of Figure 15.

Figure 22 depicts a flow chart of the Baseline Substraction algorithm of Figure 2.

Figure 23 depicts a flow chart of the Pre-Processing Begin/End Detection of Figure 1.

IV. Detailed Description of the Preferred Embodiment

The present invention provides a method and apparatus for detecting and analyzing information-containing signals in chromatographic data. In the preferred embodiment, the invention analyzes chromatographic data from DNA sequence analysis machines employing various and sundry imaging techniques, including autoradiograms, four lane-single fluor, and single lane-four fluor data. The invention further includes general and dedicated apparatuses for performing the invented method. Finally, the invention also includes a kit comprising one or more of the following components in combination with the invented method: a DNA sequence apparatus, signal detection apparatus, information storage devices for preserving chromatographic data before, during and after analysis, and output devices for displaying the analyzed sequence information.

For DNA sequence analysis, the invented method takes as input the output from a DNA sequencing apparatus and returns the called sequence, aligned traces, and band metrics for each called base. After each sample segment is read, its called sequence, aligned traces, and band metrics are joined to previous read segments. After an entire ladder has been read, a final step analyzes each called base's metrics and assigns a quality value. The quality values are used to identify the largest block of high quality sequence and establish left and right cutoff values. If a "preamble" sequence is available, the base calling software will attempt to locate the preamble in the called sequence and set the left cutoff value beyond it. Such preamble sequences may include primer sequences or known sequences which are to be excluded from the collected data. This latter step improves the chance that the sequence called by this software would merge with the least amount of human intervention.

The following sections provide a detailed description of each function of the invented method. The illustrative embodiments of the invention exemplify the application of the useful characteristics discussed below, and further reference to these and other useful and

novel features is made in the following discussion of each illustrative embodiment. These exemplary embodiments are intended to limit neither the scope of the method and apparatus that are needed for performing the invented method.

Referring to Figure 1, the invented base calling software first performs a preprocessing step 102 on the input data set 101. Preprocessing can include spectral separation, background subtraction and interpolation of input data set 101. The preprocessed data set 103 then enters Steps 104-106, which reads successive sample segments of the preprocessed data. The sample segments 104 may be any suitable size which provides efficient signal analysis. In the most preferred embodiment of the invention, the first segment is 2048 scanline samples. Subsequent segments are also 2048 samples, with 148 samples overlapping the previous segment. The following description is based on the most preferred sample segment size, although the scope of the invention is not intended to be limited to that segment size.

Each sample segment 104 is first analyzed to estimate the coarse band spacing. Subsequently, the segment 104 is analyzed at second time 106 to refine the predicted band spacing. The band spacing drives the selection of the reconstruction filter employed during blind deconvolution. Band spacing and filter band width are inversely related. Once a sample segment of 2048 scanlines is read twice (a refined sample segment) and its band spacing measured and normalized for that 2048 scanline segment, the next sample segment of 2048 samples is read. The next sample segment overlaps the previous segment by 148 scanlines (or about 15 nucleotide bases) to establish the frame and relative positioning of adjacent segments. Subsequent segments 104 are similarly processed until the final sample segment 104 is reached. If fewer than 2048 scanlines are available in the original data set, then pseudo-random noise is generated to fill the sample segment to the required 2048 samples. Pseudo-random noise is preferred because sources of non-random noise will cause improper processing during the blind deconvolution and alignment steps.

Once all sample segments have been processed (read twice, normalized and the segments aligned), the processed and aligned data is analyzed in three fuzzy logic blocks. Fuzzy logic allows multivalued logic to enhance peak detection. By using fuzzy logic, a gap is "somewhat big," a band is "not so tall." Fuzzy logic also provides logic operators (for example AND, OR, NOT). Each fuzzy logic block in the base calling method provides a

particular analysis of its data. The logic blocks operate on normalized input data and essentially classify each band based on absolute and relative criteria which are based on the band's neighboring bands. For example, fuzzy logic block 108 analyzes each base, its upstream context and assigns a quality value to each called band (base identity). Following the assignment of quality values, fuzzy logic block 108 also identifies the largest block of high quality data in the processed and aligned data 107. The right and left cutoff points for the high quality data block are recorded and set as left and right cutoff points. The output data set 109 includes the finished traces, the called bases with their assigned quality values and the suggested left and right cutoff points. Output data set 109 can optionally be visually enhanced to normalize all bands to about the same band amplitude and to remove the saw tooth appearance of the non-visually enhanced traces.

A. **Preprocessing**

Referring to Figure 2, input data set 201 is an 2048x4 trace matrix. The first step is to established Begin and End points by analyzing input data set 201 for the first scanlines containing above-background signal and for the last scanlines containing such signal (See trace 202a as an example.) Large signal spikes, due to artifacts such as primer peaks, are excluded.

Referring to Figure 23, the Preprocessing Begin/End subroutine identifies the Begin and End points based on signal amplitude. The Begin and End points define the usable signal for subsequent operations. Usable signal typically begins just left of the largest left-most signal amplitude (the so called primer peak), and continues until either the end of the sample segment data or another region of large signal amplitude is encountered. (The latter peak is typically called a biostreptation peak.) More specifically, steps 2302 through 2305 identify the putative start and end points by breaking the sample segment into zones and determining the maximum signal amplitude in each zone. Step 2306 determines whether a second primer peak is present. If the second peak is present, Step 2306 sets the Begin point at the second primer peak. Steps 2307 and 2308 make final adjustments to the Begin and End points, setting the Begin point to the first sample with amplitude below the mean of the first half of the signal, and setting the End point back 350 samples from end.

Referring to Figure 2, the baseline of the Preprocessed Begin/End data 202 is then determined at Step 203. A single baseline is established for each fluor of the Preprocessed Begin/End data 202. The baseline is subtracted from Preprocessed Begin/End data 202 to generate a baseline subtracted data set 203. For example, after baseline subtraction, the localized data set 207 becomes baseline subtracted data set 208. In a more preferred embodiment of the invention, a single baseline is established based on data from all lanes. This best determines the baseline beneath a run of poorly resolved bases in one lane. Currently available DNA sequence data precludes this embodiment because no two fluors reliably a common baseline.

Referring to Figure 22, the baseline can be established by estimating the baseline of the Preprocessed Begin/End 2201. In the working embodiment, each trace lane is processed twice using a rising exponential threshold. One pass is made from left to right (establishing one baseline) (Step 2202), and the next pass is from right to left (establishing another baseline) (Step 2203). By taking the geometric mean of the two baseline approximations a fairly natural subtrahend is produced. (See sample traces 2205 and 2206.)

To establish a baseline approximation using a rising exponential threshold, a threshold is initially set to the lowest point found within the first 10 samples. As each successive sample is considered, the threshold is incremented by an exponential which slowly ramps upward. When a subthreshold sample is encountered, the baseline between the previous subthreshold point and the current point is taken to be a line segment between the points. The threshold is reset to the new subthreshold sample value and the process continues. If, after 100 samples no subthreshold point has been found, a 100 point segment of the baseline is computed (again, piecewise linear), and the rate of rise of the exponential is increased. The exponential is calculated to rise by 1/3 the amplitude of the most recent subthreshold point over a span of 75 samples.

Following baseline subtraction, baseline subtracted data set 203 is preferably spectrally or leakage separated. This step markedly improves the quality of capillary electrophoresis data. For slab gel data with a signal to noise ratio of 2.0 or less, separation step 204 significantly improves data quality, such that unreadable data can become readable. The separation step 204 is preferably performed during preprocessing without user input.

For four fluor-single lane data, the baseline-subtracted data set 203 is spectrally separated. For single fluor-four lane data, data set 203 is leakage separated. For either separation, the separation algorithm 204 builds a characteristic matrix (CHM) which is used to perform the separation. For spectral separation, the characteristic matrix captures the spectral cross-talk ratios in four fluor data. For leakage separation, the characteristic matrix is generated from the ratio of leakage from the signal in the center of the lane in question to the signal in adjacent lanes. For capillary electrophoresis data, the ratios are measured at "peak center." For slab gel data, all data points are used to generate the characteristic matrix.

A separation matrix is calculated according to the formula,

$$SST = inv(CHM*CHM^T)^{-1}$$

where the columns of CHM hold the ratios for each respective lane. The ratios are normalized so that the largest element in each column has a value of one. The result of separation 204 is a separated data set 204.

Processing steps 104-106 are optimally performed on preprocessed data 206 containing at least 8 scanlines (samples) per band. To increase the number of scanlines per band, a baseline subtracted data set 203 or a separated data set 204 may optionally be enhanced to double or triple the number of samples using cubic spline interpolation 205.

B. Reading

Referring to Figure 3, the exemplified reading step analyzes sample segments 301 of 2048 scanlines. Each sample segment 301 first undergoes blind deconvolution 302 to cancel the effects of an unknown laurentian blurring function and to normalize the amplitudes of the traces. Blind deconvolution is described in the U.S. Patent to T.G. Stockham and J.T. Ives (No. 5,273,632), which is incorporated by reference herein.

The presently invented method includes the following improvements over the method of Stockman and Ives. The first 2048 samples are blind deconvolved with an initial narrow-band guess for the filter band width ("FBW") value. The narrow-band guess is made so that the initial reading does not overestimate the band density along the sample segment. Given the resulting conservative estimate of the band density, a subsequent, more apt FBW is chosen and the segment is reread using it. The FBW chosen for the second read of each segment also serves as the FBW used for the 1st read of following segment. This iterative

approach to determining the best FBW has proven invaluable in practice; the band densities may vary from about 6 samples/band to about 50 samples/band, and do so not only within a given ladder but also from sequencing run to sequencing run. In the preferred embodiment of the invention, the method is adaptable to a wide range of acceptable inputs.

The invented method includes a means for selecting the filter band width (FBW) during blind deconvolution 302. In the working embodiment median band spacing is mapped to a filter band width value using the following equations:

$$K = 2 * \text{sqrt}(\ln(0.23)/-0.5)$$

$$FBW = 0.5 + (K/\text{med\_band\_spacing}) * (2048/(2*\pi))$$

The blind deconvolution step 302 deblurs the signal and normalizes its amplitude. Following blind deconvolution, an extra-normalization function 303 adjusts band spacing due to mobility differences in the samples. Extra-normalization 303 also corrects for the tendency of blind deconvolution to create spurious bands, especially in regions of mono-, di- or tri-nucleotide repeats where one or more lanes are band-lite for extended regions.

Referring to Figure 4, extra-normalization 303 corrects two types of artifacts created by blind deconvolution. Path 406-410 cancels artifacts created in band-lite lanes. Briefly, the blindly deconvolved data set 406 is scanned for band-lite lanes by comparing the relative lane signal strengths and the relative lane band frequencies. The proxy used for the lane signal strength analysis is the 97th percentile signal amplitude found in each lane. The proxy used for the band frequency is the proportion of the signal over which the lane in question has the largest signal amplitude. If a lane has less than 15% of the total bands found in all four lanes in a sample segment, and if the band amplitudes are low relative to the other lanes, the amplitudes of the bands in that lane are attenuated. If the band amplitude is lowest, those amplitudes are attenuated by one-half. If the band amplitude is above the lowest, the amplitudes are attenuated to three-quarters of the original band amplitude. In contrast, in ideal sequence data, where A, G, C, and T are equal in frequency, each trace should dominate 25% of the time.

Extra-normalization path 401-403 corrects for mobility differences between lanes and performs the actual band-lite attenuation 404. Briefly, the blind deconvolved data set 401 is analyzed to identify any regions with inordinately large or coincident bands. These regions

are set to zero (base-line). If these regions were not set to zero the Monte Carlo alignment algorithm 403 would produce an aberrant alignment which focused on separating them.

Referring to Figure 14, mobility shifts are most noticeable in the autoradiogram and single fluor-four lane data particularly near the edges of the gel. The prior method of Stockham and Ives described an algorithm which attempted to align the lanes by driving the band spacing to as nearly a uniform value as possible. This approach was limited because, without proper alignment, many true bands would go undetected because they were shadowed by other bands. The algorithm attempted to normalize spacing between detected bands, yet the algorithm knew of only a simple majority of the bands present in the data.

The present invention uses an algorithm which does not use band position or spacing information. Instead, the present invention seeks to maximize the integral of the "envelope" of all four lanes of data when they share a common baseline. Alignment is accomplished using a Monte Carlo search of a 3D space, where the x-axis defines the relation between the A and G lanes, the y-axis defines the relation between the AG relation and the C lane, and the z-axis defines the relation between the AGC relation and the T lane. An initial set of possible alignments are chosen, each triple is applied to the traces to be aligned, and the integral of the resulting envelope is calculated. A subset of the triples, those yielding the largest integrals, are then refined. The triple which yields the lowest integral is removed from the set under consideration. It is replaced by a triple which results from a random alteration of the triple which yields the largest integral. When either a maximum number of iterations has occurred or the variation within the set of high integral triples has reached a suitably low value, the highest yielding triple is chosen as the alignment vector for the segment under consideration.

More specifically, the search is conducted in a three dimensional space, where the x-axis specifies the offset between $trace_1$ and $trace_2$, the y-axis specifies the offset between the $trace_{1,2}$ registry and $trace_3$, and the z-axis specifies the offset between the $trace_{1,2,3}$ registry and $trace_4$ (See illustration 1401). The algorithm employed was originally described by W.L.Price in *The Computer Journal, Vol. 20, No. 4*, which is incorporated by reference herein.

Initially, a set of putative alignment solutions 1401 is generated. The addresses of the lattice points of 6 concentric cubes centered about a point in the space are used as the initial alignment solutions. The first time the procedure is used the central point of the concentric

cube lattice is the origin (x_0=0,y_0=0,z_0=0). Subsequent calls can either continue to center the lattice on the origin, or they can bias the search by centering the lattice on the previous alignment solution (x_{n-1},y_{n-1},z_{n-1}).

Each alignment guess is converted into a shift vector of four values, wherein one value is 0. Each trace in the matrix is shifted by the amount specified in the shift vector, the envelope of the shifted traces is obtained (the maximum value of the four trace values found at each position along the traces), and is summed. The sum represents the integral of the envelope produced by the alignment guess. A low integral value represents a poor alignment (see, e.g. illustration 1402, where the bands are aligned behind others, not arranged "shoulder to shoulder"), whereas a high integral value corresponds to a good alignment (see, e.g. illustration 1407, where all bands are fully exposed, arranged "shoulder to shoulder").

Once all alignment guesses have been evaluated, the worst alignment solution is replaced by a small, random perturbation of the best alignment solution 1405. The new alignment solution is evaluated, and the process repeats, replacing the new worst alignment with a perturbation of the new best alignment. Eventually, the set of points in the 3D space converge about the best alignment solution 1406.

Referring to Figure 3, following extra-normalization 303, Step 304, peak detection and refinement, occurs. The aligned traces then undergo putative peak detection. Referring to Figure 5, putative peak detection 502 is performed on the blind deconvolved, extra-normalized data set 501 (unstopped, attenuated and with the relative mobilities corrected). A trace envelope is first determined. The Stockham and Ives Patent described detecting peaks in each trace separately with thresholds derived from the underlying data. In the invented method, the trace envelope is peak-detected and no thresholds are employed. A peak is liberally defined to be a sample which is taller than either of its two neighbors. Subsequent processing culls this liberally defined putative peak list. This form of peak detection is both faster (one trace instead of four) and less prone to error (no subthreshold peaks). In contrast, the Stockham and Ives Patent required individual trace peak detection because its alignment algorithm attempted to determine lane alignment using peak location information.

To identify errors in putative band detection 502, including insertion errors, each putative peak's instantaneous spacing, cross banding, height, and spacing to adjacent bands is measured (Step 503). These observed band spacing measurements are fit with a quadratic

curve. This quadratic fit is used as the expectation of the band spacing along the entire read segment. This approach to defining the expected band spacing is sufficiently general to handle segments where, as in the Stockham and Ives Patent, the average spacing is an adequate expectation, as well as segments where the spacing changes radically. In the invented method, more information was found necessary to sufficiently identify insertions and regions of deletions, and as a result, the invented method can resolve a series of insertions and deletions.

The first of three fuzzy logic blocks 504, OmitOkN Fuzzy Logic, is then used to identify bands which are most likely insertion artifacts of the band detection process. This block classifies the detections as OK, AMBIGUOUS or OMIT. The putative bands given the OMIT classification are removed from the putative peak set. Referring to Figure 6, each band has several of its attributes 601 examined by this first logic block. If a band is where it ought to be with respect to either of its neighbors, then variable okSp is set "TRUE" (Step 602).

Referring to Figure 7, the intent of the membership function for the OmitOkN Ok Spacing fuzzy logic block is to "accept" a spacing measurement which is an integer multiple of the expected spacing. Consequently, the observed spacing is normalized to a value on the interval [0..1] using its relationship to expected spacing (Step 702). In the example given in block 702, the normalized spacing of 0.3 is found to be OK with a truth value of 0.7 (Step 703 and Example 704). Given the vagaries of band migration, compressions, band shape (hence band peak position), and other factors, a peak spaced 17 from its neighbor when the expected spacing is 13 is neither ideal nor terrible.

Referring to Figure 8, for the OmitOkN Bad Spacing fuzzy logic block, the intent of the membership function is to "deprecate" a spacing measurement which is not an integer multiple of the expected spacing. Consequently, the observed spacing is normalized to interval [0..1] using its relationship to expected spacing (Step 802). In the example given in Step 802-03 and Example 804, a normalized spacing of 0.3 is found to be BAD with a truth value of 0.5.; this spacing is not as good as it could be.

If a band is not where it ought to be with respect to either of its neighbors then variable abSp is set "TRUE" (Step 603). If the amount of "cross banding" (i.e. the amount of competition by two bands for a particular region of the read segment) is high, then variable badXb is set "TRUE" (Step 604). Similarly, if there is negligible cross banding then variable

negIXb is set "TRUE". Referring to Figure 9, cross banding designates the amount of competition for the scanlines underlying a detected band. Bands of a dubious nature have wide ranging cross band ratios due to their apex's proximity to the baseline. However, compressions and stops, with significant amplitudes, can have their cross banding measured. The cross banding membership function is best used in identifying OK or AMBIGUOUS bands. In the diagram provided (Example 901), the first complex has two bands vying for the same location, with the second largest band having one-half the amplitude of the largest. The cross banding ratio (Step 902) is the amplitude of the largest band divided by the amplitude of the next largest band, or in this case Xb=2.0. In the second complex, where one band is clearly the band of choice, this ratio approaches infinity. In the example given in Step 903, with a cross banding ratio of 1.5, the badXb membership is 0.25, while the negligibleXb membership is 1.0; in other words, while a ratio of 1.5 is found negligible, the band legitimacy will be questioned.

The band height is also categorized as either tiny or ok (Step 605). Referring to Figure 10, for the height membership functions the membership sets are best customized for the general signal quality one observes from the machine providing the data. In the working embodiment a function of the median value of amplitudes measured where bands intersect determines the height membership function break points. In particular, the tinyHt function breaks at 0.4*med_intersect_pt and is zero by 1.1*med_intersect_pt. Similarly, the okHt function comes off zero at 0.5*med_intersect_pt and flattens off at 1.0 at 1.5*med_intersect_pt. The blind deconvolution process normalizes band amplitudes to interval [0..1], with most bands having a height in excess of 0.1. This example given is typical in that it begins deprecating a band based on its height when the height falls below 0.07. In the example given in Step 1002, the measured band height is 0.1 and has membership in okHt of 1.0 and in tinyHt of 0.0. The band has, per this example of the sets, sufficient height.

These six variables then serve as input to fuzzy combinational logic. A significant advantage of fuzzy logic is that it works with and can resolve contradictions among rules involving these variables. Bands classified as OK are those with negligible cross banding and either OK height or OK spacing (Step 606). Bands classified as ambiguous exhibit bad cross banding and either OK height or OK spacing and little height (Step 607). Ambiguous bands

are typically those where the band is correctly positioned with sufficient amplitude but significant cross banding (Step 607). Bands classified as clear insertions, and therefore to be omitted, are characterized by negligible height (Step 608). Cross banding is not considered when deciding whether a band should be omitted because usually insertions are made very close to the baseline where cross banding measurements are unreliable.

The strength of the rule firings is then used to scale the output sets (Step 609). In the example given (illustration 610), the output set OK is scaled with amplitude 1.0, output set N (ambiguous) is scaled with 0.25, and set OMIT is scaled with 0.0. Defuzzification, or obtaining a crisp (conclusion) value from the output rule sets, is achieved by calculating the centroid of the resultant "masses". In the example, the conclusion reached is that the band is OK (Step 611).

Referring to Figure 5, following fuzzy logic block 504, each peak's instantaneous spacing, instantaneous band width, spacing to its left neighbor (left spacing), band width and called bases is remeasured (Step 505). These observed band spacings are fit with a quadratic curve which then serves as the expected spacing along the read segment. Similarly, the observed band width measurements are also fit with a quadratic curve which serves as the expected band width along the read segment.

The second fuzzy logic block 506, GapCheck Fuzzy Logic, then identifies bands, or gaps between bands, where one or more bands may need to be inserted to achieve the band spacing predicted by the quadratic fit. This block classifies the detections as NORMAL, SPLIT or SUFFERING FROM UPSTREAM TURBULENCE. The gaps are split and a suitable number of bands are inserted (Step 507). The bands given the SPLIT classification are split a suitable number of times, with the division points being the centroid of the interval to be split. The centroid is used to place the insertion on the shoulder of a poorly defined band, and not in the bottom of the trough between the SPLIT band and its left neighbor. Depending upon the size interval, and the expected band spacing, one or more insertions may be made. Each insertion has a defined Begin, Middle and End scanline value.

Referring to Figure 11 for more detail, each band pair considered by fuzzy logic block GapCheck has several attributes which are measured (Step 1101). In particular, the expected spacing curve, expected width curve, band width, left gap (gap to the leftmost neighbor) and sequence is determined. The upstream sequence is assigned a measure of GC-richness (Step

1102). These measurements, coupled with the GC richness of the sequence of the 5 bands to the left, are informative in identifying bands which need additional bands added to their left. The gap is normalized with respect to the expected spacing onto interval [-1..inf] (Step 1103). Unlike the OmitOkN logic, where the logic determines if a band is located where it should be (independent of its absolute spacing but focusing instead on how far off the spacing curve it is), in the GapCheck logic block, the concern is on the absolute distance of the band from its left neighbor. If the gap is an integer multiple of the spacing curve (say three spaces from its left neighbor) two bands are inserted to its left to establish the proper spacing. In addition to the gap between bands in the pair, this logic also considers the widths of the bands. Band width is normalized onto interval [-1..inf] (Step 1104). Usually, when band resolution decreases and a region in the observed trace contains fewer peaks than are required, one or both bands in the pair is wider than it should be. The gap between the bands can be marginal and the band width can be the determining factor. Finally, large gaps and band widths should be viewed less aggressively in the presence of upstream GC-richness.

The normalized left-gaps of each band in a band pair are classified as big (Step 1105), medium (Step 1106) or small (Step 1107). Figure 12 provides details of the GapCheck band gap membership function. Briefly, the membership function characterizes an observed gap measurement (ogp) if it differs from expectation (egp). The gap is measured between B_n and B_{n-1} (Step 1201). The observed gap is normalized to interval [-1..inf] with the equation: ngp = ogp/egp - 1.0. Referring to the example in Step 1203, a normalized gap of 0.1 is found to have 0.0 membership in all sets; that is, the gap meets expectations and is neither small, medium nor big (Step 1203).

Referring to Figure 11, in Step 1108 the normalized widths of each band in a band pair are classified as big. Figure 13 provides details of the GapCheck band width membership function. This membership function characterizes an observed band width measurement (owd) if it exceeds expectations (ewd). The width is measured between B_n's Begin and End points (Step 1301). In the example given in Step 1303, a normalized gap of 0.2 is found to have membership in BigWidth of 0.2; the band is not that wide, but it is wider than expected (Step 1304).

Referring to Figure 11, in RULE NORM (Step 1109), band$_n$ is not marked as needing its left gap split if any of the following are TRUE:

a) there is a large gap ($bigGap_n$) but the upstream sequence is GC-rich, or

b) the gap to the first band in the pair is small ($smlGap_{n-1}$) and the two bands are not wide ($!bigWid_n$ and $!bigWid_{n-1}$) (i.e., ignore the gap between the bands), or

c) the gap between the two bands is not large ($!bigGap_n$).

In step 1110, RULE SPLIT marks $band_n$ as needing its left gap split if either of the following are TRUE:

a) the gap between the two bands is large ($bigGap_n$) and the first and/or second band is wide ($bigWid_{n-1}$ or $bigWid_n$) or,

b) the gap between the two bands is large ($bigGap_n$) and the gap left of the first band is not small ($!smlGap_{n-1}$) and the upstream sequence is not gc-rich (!gcrich).

RULE SPLIT (a) detects the combination of a wide and normal band (in either order) while RULE SPLIT (b) selects a run of wide bands separated by large gaps. The strength of the rule firings is then used to scale the output sets (Step 1111). In example 1112, the output set NORMAL is scaled with amplitude 1.0 and output set SPLIT is scaled with 0.25. The conclusion is formed by calculating the centroid of the resultant "masses". In example 1112, the conclusion reached is that the band is NORMAL.

To identify putative peak insertion errors, Step 507 remeasures the cross banding, instantaneous spacing, band height, band amplitude, and the spacing (left and right gaps) to adjacent bands. The observed band spacing measurements are fit with a quadratic curve. This quadratic fit is used as the expectation of band spacing along the entire read segment. The OmitOkN Fuzzy logic block (Step 508) is then used to identify bands which are most likely insertion artifacts of the band detection process. Any and all such bands are removed from the putative peak set. Newly proposed insertions may be deleted in this step. The fuzzy logic band refinement stage adds the important advantage of reducing insertions and deletions and preventing arbitrary band calling when the reader encounters two or three base regions of signal dropout. See Figure 6 and the accompanying text for details of insertion detection. The set of putative peaks which survive this processing are recorded as the bands for the read segment under consideration (Step 509).

C. **Processing and Alignment**

Referring to Figure 1, reading function 104-106 consecutively processes sample segments until all of the input data set 101 is analyzed. Because each sample segment 104 overlaps the previous sample segment by a predetermined amount, the relative positioning of each read and aligned sample segment 106 is known. Step 107 assembles all of the read and aligned sample segments 106 to form a processed and reassembled sample segment 107.

D. Post-Processing Editing

In the working embodiment of the invention, a final process analyzes the set of measured band features with a third fuzzy logic block, BaseQual fuzzy logic block 109. This block assigns a quality measure to each called band. This block evaluates each band based on the band height, width, shape, left and right gap, cross-banding and baseline "buzz." This quality value, on the interval (0.0 to 1.0) can be used during subsequent sequence alignment/merging steps. The present invention uses the quality value to select the longest block of high quality sequence to be considered for alignment and merging with other sequences into a large DNA sequence. The algorithm that selects the left and right cutoff points generates a surface, with the x-axis labeled MOVING AVERAGE FILTER WIDTH, the y-axis labeled THRESHOLD, and the z-axis labeled READ LENGTH. The quality values are filtered with six moving average filters, and the filtered data is compared against nine thresholds. The longest contiguous block of above threshold filtered quality values provides the read length value for the surface for a particular filter width, threshold pair. Finally, this surface is scaled so that narrow filter and high threshold read lengths are favored over wide filter and low threshold read lengths. The surface maximum z-value is then chosen as the read length, and the associated first and last above threshold filtered quality value indexes serve as the left and right cutoff points, respectively. If a "preamble" sequence was submitted to this EDIT stage, and if the sequence is found beyond the established left cutoff point, the cutoff point is moved further left to exclude the "preamble" sequence.

Referring to Figure 15, the BaseQual fuzzy logic algorithm assesses the quality of the called bases. Experience has shown that some sequence assembly algorithms fail to assemble sequences containing regions of incorrect sequence and that others can only succeed when each base is accompanied by an indication of its quality (or inversely, its probability of error). In the former case, if the incorrect sequence regions are masked from consideration by the

assembly program, the bulk of the good sequence will successfully assemble. In the latter case, if the low quality regions are identified, the overall base caller product will assemble. In either case, incorrect sequence, encountered in isolation, can be and usually is identified by an experienced technician using visual inspection. That process is time consuming and monotonous and subtle errors may go undetected. In general though, incorrect base calling is done where the underlying data traces are marginal.

The BaseQual routine, automates quality assessment by measuring and analyzing multiple features (Step 1501) of each called base. Fuzzy logic is used to identify certain band presentation patterns and assign levels of quality to them. These band features include the band height, cross-banding, band width, band shape, the band's small gap and the band's large gap.

Band height variations are informative in many of the classifications. Six fuzzy variables are used to classify a band's height (tiny, small, moderate, normal, tall and collectively, OK) (Step 1502). Referring to Figure 16 for details of the BaseQual height membership functions, the membership function characterizes an observed band height measurement. A band with a "tiny" or "small" height is usually suspect, with the tiny bands being more suspect that the merely small. Moderate height bands, and tall bands, also require scrutiny. Tall bands are suspect because usually they are found amid stops, compressions, and, on slab gels, artifacts. In the example given in Step 1602, a band height of 0.18 is found to have membership in NormalHeight of 1.0, which is to say that the band's height is within tolerances (See Example 1603).

Referring to Step 1503, cross banding, the measure of competition by two traces for the same region of the trace, is also informative. Referring to Figure 17, the BaseQual cross banding membership functions characterize an observed band's cross banding measurement. The cross banding measurement is the ratio of the dominant trace to the next dominant trace. Ratios above 1.5 are deemed to have negligible cross banding, whereas those with lower ratios (with 1.0 being the lowest ratio possible) are suspect. Referring to the example in Figure 1702, a cross banding ratio of 1.35 is found to have membership in negligibleXb of 0.33 (and 0.67 in the negation, !negligibleXb). Referring to Step 1504, band width (normalized based on a quadratic fit of observed band widths), is another informative variable. In Figure 18, the BaseQual band width membership function, a band's observed

width is normalized with respect to the expected band width. The intent of the membership function is to determine how normal the normalized band width is. Referring to the example in Step 1802, a normalized width of 0.2 has membership in the Normal set of 1.0 (See Example 1803). Referring to Step 1505, the band shape, the linear correlation coefficient between the coefficients of a quadratic fit of the band and the coefficients of a quadratic fit of an ideal band, identifies abnormally shaped bands. The BaseQual band shape membership function is informative in determining the quality of the base call. The range of band heights and widths observed in a run varies considerably. In one embodiment, sample rate conversion normalizes the observed band width, the band amplitude was normalized to 1.0, and the result was then compared against an ideal, gaussian bell-shaped band. The approach is computationally expensive and much information regarding the observed shape is discarded through the morphing process.

In a more preferred embodiment, each band's height values are fit with a quadratic curve. Similarly, an ideal band shape is fit with a quadratic curve. (The ideal band shape is defined to have normal height and the expected width.) This approach reduces each sample set to an equal number of points. The shape metric is taken as the linear correlation coefficient of these two sample sets. Experience has shown that a band's shape is "abnormal" if this shape metric falls below 0.5. Referring to the example in Step 1902, a shape metric of 0.6 is found to have membership in GoodShape of 1.0 (See example 1903).

Referring to Step 1506, "baseline buzz," defined as the ratio of two other ratios, helps identify regions of the trace (usually the ends) where there is competition by several traces for the called band's domain. Toward the margins of a trace the baseline can often become quite busy, and when it does the quality of the underlying data, and the reads made thereon, become suspect. Baseline buzz can result from either incorrect signal processing, or from underlying data being so erratic as to defy correct signal processing. In either case the called sequence should come under suspicion. Referring to Figure 20, a buzz measurement above 0.2 begins to signal a problematic sequence. Referring to the example is step 2003 and Example 2004, a buzz measurement of 0.28 has membership in okBuz of 0.63 (and 0.37 in the negation, !okBuz). In this case, the band's quality has come into question.

Referring to Step 1507, the gaps to a band's left and right neighbor are further informative variables in assessing band quality. These measurements help identify bands

that, despite all previous efforts to the contrary, remain positioned too close or too far from a preferred position. Referring to Figure 21 for details of BaseQual band spacing membership functions, a band's observed spacing is normalized with respect to the expected spacing. The intent of the membership function is to determine how normal the normalized band spacing is. In the example given in blocks 2102 and 2103, a normalized spacing of 0.2 receives an unqualified OK, with membership in OK Spacing of 1.0.

Referring to Step 1508, logical combinations of several variables (e.g. buzz, width, shape, and spacing) help keep the rules for assigning a quality value to a band tractable. Variable *1Bad* indicates that one of the measures was out of tolerance. Similarly, variables *2Bad*, *3Bad*, and *4Bad* indicate that two, three, or all four measurements are out of tolerance. Finally, variable *4Ok* notes that all four measurements are in tolerance.

Subsequently, a quality assessment is determined through application of a series of nine rules. In RULE QUAL10, the lowest quality assessment is made of bands that are tiny in height and are incorrectly positioned (Step 1509). For a band which matches this rule to some degree, the rule will assign a nonzero scale value to the output set with centroid near 0 (Step 1519). The second quality assessment, RULE QUAL20, is made of bands that are short, show signs of cross banding, and are incorrectly positioned (Step 1510). For a band which matches this rule to some degree, the rule will assign a nonzero scale value to the output set with centroid near 13 (Step 1519).

The third quality assessment, RULE QUAL30, is made of bands which are tiny in height yet correctly positioned (Step 1511). A band which matches this rule to some degree will be assigned a nonzero scale value to the output set with centroid near 25 (Step 1519). The fourth quality assessment, RULE QUAL40, is made of bands with small or moderate height and which show signs of cross banding or have 3Bad or 4Bad attributes (Step 1512). A band which matches this rule to some degree will be assigned a nonzero scale value to the output set with centroid near 38 (Step 1519).

The fifth quality assessment, RULE QUAL50, is made of bands with small or moderate height and which show either some degree of cross banding or have 2Bad or 3Bad attributes (slightly better than quality class 4 in that these might have one less bad attribute) (Step 1513). A band which matches this rule to some degree will assign a nonzero scale value to the output set with centroid near 50 (Step 1519). RULE QUAL60, the sixth quality

assessment, is applied to bands with OK height but which show signs of baseline buzz, non-negligible cross banding, or have 2Bad attributes (Step 1514). A band which matches this rule to some degree will assign a nonzero scale value to the output set with centroid near 63 (1519).

Bands which have higher degrees of quality satisfy the seventh to ninth quality assessments. The seventh quality assessment, RULE QUAL70, is made of bands in one of three general classes. (Step 1515). One class of bands has OK height, little baseline buzz, negligible cross banding, but has 2Bad attributes. Another class of bands shows negligible cross banding, has OK height, no baseline buzz, is correctly positioned, but has both abnormal width and bad shape. (This class, named *runfil*, is characteristic of a band inserted within a poorly resolved run of bands). The final class of bands has 4Ok attributes but has small height and possibly some degree of cross banding present. A band which matches this rule to some degree will assign a nonzero scale value to the output set with centroid near 75. (Step 1519).

The eighth quality assessment, RULE QUAL80, is made of bands with OK height, little baseline buzz, negligible cross banding, but 1Bad attribute. (Step 1516). A band which matches this rule to some degree (many do) will assign a nonzero scale value to the output set with centroid near 88. (Example 1519). The top quality assessment, RULE QUAL90, is made of bands with absolutely nothing visually wrong with them (Step 1517). A band which matches this rule to some degree (again, given good quality input, many do) will assign a nonzero scale value to the output set with centroid near 100 (Step 1519). Finally, as with all the other fuzzy logic blocks, the output sets are scaled with the strength of their respective rule firings, and the centroid is calculated to determine the final quality assessment.

E. Final Sequence Assembly

The final quality assessments from the BaseQual Fuzzy Logic analysis control the length of the final sequence 109. Where high quality sequence data is desired, the quality assessments determines may limit the read length of the final sequence. Where longer read lengths are desired, and lower quality sequence is acceptable, the quality assessments can aid is correlating the resulting sequence data from other sequence analysis. For example, when overlapping sequences are obtained, the quality assessments can determine which base calls

are more reliable. Similarly, when both strands of a DNA sequence are available, the quality assessments aid in identifying higher probability base calls.

F. Computer Implementation of the Base Calling Software

The invented Base Calling Software can be implemented on standard desktop computers, such as Pentium- and 486-containing PC's. Computers with less powerful processors are also suitable, although the overall processing time for each input data set will be slower. Such computers will preferably include at least a central processing unit, dynamic memory and a device for outputting processed information. The invented base calling software can be stored on any suitable storage media, including computer diskettes, removable media, hard-drives, CD's, magnetic tapes and similar electronic storage means.

```
/************************************************************************
 * FILE: AboutBQ.hxx
 * AUTHOR:  Andy Marks
 */
#if !defined(_ABOUTBQ_HXX_)
# define _ABOUTBQ_HXX_
#if defined(WIN32)
class _declspec( dllexport ) AboutBQ
#else
class AboutBQ
#endif
{
public:
   AboutBQ();
   ~AboutBQ();
   char const* productName() const;
   float      productVersion() const;
   int        numProcedures() const;
   char const* procedureName(int pdx) const;
};
#endif

/************************************************************************
 * FILE:      AboutBQ.cxx
 * AUTHOR:    Andy Marks
 */
#include <basecall/AboutBQ.hxx>
static char const* plut_[] =
{
 "Base Calling With Quality Metrics",
 "Set Spectral Separation Matrix",
```

```
  "Set Read Start And End Scanlines",
  "Illegal Procedure Index"
};
#define NPS (sizeof(plut_)/sizeof(plut_[0]))
AboutBQ::AboutBQ()
{
}
AboutBQ::~AboutBQ()
{
}
float
AboutBQ::productVersion() const
{
  return 0.995f;
}
char const*
AboutBQ::productName() const
{
  static char const* prod = "BaseQual: Cimarron Software Inc.";
  return prod;
}
int
AboutBQ::numProcedures() const
{
  return NPS-1;
}
char const*
AboutBQ::procedureName(int idx) const
{
  if(idx<0 || idx>=NPS)
    idx = NPS-1;
```

```
  return plut_[idx];
}

/*********************************************************************
 * FILE: bandqual.cxx
 */
#include <basecall/mb.hxx>
#include <basecall/fuzzyset.h>
void
RdrOut::bandqual()
{
 static double const tinyh_x[] = {0.00, 0.02, 0.03},
                     tinyh_y[] = {1.00, 1.00, 0.00};
 static double const smalh_x[] = {0.015, 0.025, 0.04, 0.05},
                     smalh_y[] = {0.000, 1.000, 1.00, 0.00};
 static double const modrh_x[] = {0.035, 0.045, 0.14, 0.16},
                     modrh_y[] = {0.000, 1.000, 1.00, 0.00};
 static double const normh_x[] = {0.135, 0.145, 0.48, 0.55},
                     normh_y[] = {0.000, 1.000, 1.00, 0.00};
 static double const tallh_x[] = {0.475, 0.485, 1.00},
                     tallh_y[] = {0.000, 1.000, 1.00};
# define TINYH_SZ (sizeof(tinyh_x)/sizeof(tinyh_x[0]))
# define SMALH_SZ (sizeof(smalh_x)/sizeof(smalh_x[0]))
# define MODRH_SZ (sizeof(modrh_x)/sizeof(modrh_x[0]))
# define NORMH_SZ (sizeof(normh_x)/sizeof(normh_x[0]))
# define TALLH_SZ (sizeof(tallh_x)/sizeof(tallh_x[0]))
 static double const presx_x[] = {1.0,  1.35},
                     presx_y[] = {1.0,  0.0};
 static double const neglx_x[] = {1.35, 1.5},
                     neglx_y[] = {0.00, 1.0};
# define PRESX_SZ (sizeof(presx_x)/sizeof(presx_x[0]))
```

```
# define NEGLX_SZ (sizeof(neglx_x)/sizeof(neglx_x[0]))
 static double const normw_x[] = {-1.0, -0.4, 0.4, 1.0},
                normw_y[] = { 0.0,  1.0, 1.0, 0.0};
# define NORMW_SZ (sizeof(normw_x)/sizeof(normw_x[0]))
 static double const goods_x[] = { 0.4, 0.5, 1.0},
                goods_y[] = { 0.0, 1.0, 1.0};
# define GOODS_SZ (sizeof(goods_x)/sizeof(goods_x[0]))
 static double const goodg_x[] = {0.3, 0.5, 0.7},
                goodg_y[] = {1.0, 0.0, 1.0};
# define GOODG_SZ (sizeof(goodg_x)/sizeof(goodg_x[0]))
 static double const okbuz_x[] = {0.0, 0.2, 0.4},
                okbuz_y[] = {1.0, 1.0, 0.0};
# define GOODB_SZ (sizeof(okbuz_x)/sizeof(okbuz_x[0]))
 static double const conc1_x[] = {-0.100, 0.010, 0.125 };
 static double const conc2_x[] = { 0.010, 0.125, 0.250 };
 static double const conc3_x[] = { 0.125, 0.250, 0.375 };
 static double const conc4_x[] = { 0.250, 0.375, 0.500 };
 static double const conc5_x[] = { 0.375, 0.500, 0.625 };
 static double const conc6_x[] = { 0.500, 0.625, 0.750 };
 static double const conc7_x[] = { 0.625, 0.750, 0.875 };
 static double const conc8_x[] = { 0.750, 0.875, 0.990 };
 static double const conc9_x[] = { 0.875, 0.990, 1.105 };
 static double const concn_y[] = {0.0, 1.0, 0.0};
# define CONCL_SZ (sizeof(conc1_x)/sizeof(conc1_x[0]))
 static double const conc_x[] = {0.0, 1.0}, conc_y[] = {0.0, 0.0};
# define CONC_SZ (sizeof(conc_x)/sizeof(conc_x[0]))
 CFuzzySet *tinyH, *smalH, *modrH, *normH, *tallH;
 CFuzzySet *presX, *neglX, *normW, *goodS, *goodG, *goodB;
 int NB = bandStats_.len(), idx;
 tinyH = ConstructCFuzzySet( TINYH_SZ, tinyh_x, tinyh_y, NOHEDGE );
 smalH = ConstructCFuzzySet( SMALH_SZ, smalh_x, smalh_y, NOHEDGE );
```

```
modrH = ConstructCFuzzySet( MODRH_SZ, modrh_x, modrh_y, NOHEDGE ),
normH = ConstructCFuzzySet( NORMH_SZ, normh_x, normh_y, NOHEDGE ),
tallH = ConstructCFuzzySet( TALLH_SZ, tallh_x, tallh_y, NOHEDGE );
presX = ConstructCFuzzySet( PRESX_SZ, presx_x, presx_y, NOHEDGE );
negIX = ConstructCFuzzySet( NEGLX_SZ, neglx_x, neglx_y, NOHEDGE );
normW = ConstructCFuzzySet( NORMW_SZ, normw_x, normw_y, NOHEDGE );
goodS = ConstructCFuzzySet( GOODS_SZ, goods_x, goods_y, NOHEDGE );
goodG = ConstructCFuzzySet( GOODG_SZ, goodg_x, goodg_y, NOHEDGE );
goodB = ConstructCFuzzySet( GOODB_SZ, okbuz_x, okbuz_y, NOHEDGE );
for(idx=0;idx<NB;idx++) {
 double tinyh, smalh, modrh, normh, tallh, okbuz, runch1, runfil, fourok;
 double presx, neglx, normw, goods, goodg1,goodg2,okgp, conj, okhght;
 CFuzzySet *CONCLUSION, *C1, *C2, *C3, *C4, *C5, *C6, *C7, *C8, *C9;
 CONCLUSION = ConstructCFuzzySet( CONC_SZ, conc_x, conc_y, NOHEDGE );
 C1 = ConstructCFuzzySet( CONCL_SZ, conc1_x, concn_y, NOHEDGE );
 C2 = ConstructCFuzzySet( CONCL_SZ, conc2_x, concn_y, NOHEDGE );
 C3 = ConstructCFuzzySet( CONCL_SZ, conc3_x, concn_y, NOHEDGE );
 C4 = ConstructCFuzzySet( CONCL_SZ, conc4_x, concn_y, NOHEDGE );
 C5 = ConstructCFuzzySet( CONCL_SZ, conc5_x, concn_y, NOHEDGE );
 C6 = ConstructCFuzzySet( CONCL_SZ, conc6_x, concn_y, NOHEDGE );
 C7 = ConstructCFuzzySet( CONCL_SZ, conc7_x, concn_y, NOHEDGE );
 C8 = ConstructCFuzzySet( CONCL_SZ, conc8_x, concn_y, NOHEDGE );
 C9 = ConstructCFuzzySet( CONCL_SZ, conc9_x, concn_y, NOHEDGE );
 tinyh  = tinyH->membership( tinyH, bandStats_.hght(idx) );
 smalh  = smalH->membership( smalH, bandStats_.hght(idx) );
 modrh  = modrH->membership( modrH, bandStats_.hght(idx) );
 normh  = normH->membership( normH, bandStats_.hght(idx) );
 tallh  = tallH->membership( tallH, bandStats_.hght(idx) );
 presx  = presX->membership( presX, bandStats_.xbnd(idx) );
 neglx  = neglX->membership( neglX, bandStats_.xbnd(idx) );
 normw  = normW->membership( normW, bandStats_.widt(idx) );
```

```
goods  = goodS->membership( goodS, bandStats_.shap(idx) );
okbuz  = goodB->membership( goodB, bandStats_.buzz(idx) );
goodg1 = goodG->membership( goodG, bandStats_.sgap(idx) );
goodg2 = goodG->membership( goodG, bandStats_.lgap(idx) );
okgp   = AND(goodg1,goodg2);
C1->scale( C1, AND( tinyh, NOT(okgp)) );
C2->scale( C2, AND( smalh, AND(presx,NOT(okgp))) );
C3->scale( C3, AND( tinyh, okgp) );
okhght = OR(modrh,OR(normh,tallh));
runch1 = AND(NOT(goods),okhght);
runfil = AND(runch1,AND(negix,AND(normw,AND(okbuz,okgp))));
fourok = AND(okbuz,AND(normw,AND(goods,okgp)));
double cj1,cj2,cj3,cj4,cj5,cj6, _1bad, _2bad, _3bad, _4bad;
cj1 = AND(okbuz,AND(normw,AND(goods,NOT(okgp))));
cj2 = AND(okbuz,AND(normw,AND(NOT(goods),okgp)));
cj3 = AND(okbuz,AND(NOT(normw),AND(goods,okgp)));
cj4 = AND(NOT(okbuz),AND(normw,AND(goods,okgp)));
_1bad = OR(cj1,OR(cj2,OR(cj3,cj4)));
cj1 = AND(okbuz,AND(normw,AND(NOT(goods),NOT(okgp))));
cj2 = AND(okbuz,AND(goods,AND(NOT(normw),NOT(okgp))));
cj3 = AND(okbuz,AND(okgp,AND(NOT(normw),NOT(goods))));
cj4 = AND(normw,AND(goods,AND(NOT(okbuz),NOT(okgp))));
cj5 = AND(normw,AND(okgp,AND(NOT(okbuz),NOT(goods))));
cj6 = AND(goods,AND(okgp,AND(NOT(okbuz),NOT(normw))));
_2bad = OR(cj1,OR(cj2,OR(cj3,OR(cj4,OR(cj5,cj6)))));
cj1 = AND(okbuz,AND(NOT(normw),AND(NOT(goods),NOT(okgp))));
cj2 = AND(normw,AND(NOT(okbuz),AND(NOT(goods),NOT(okgp))));
cj3 = AND(goods,AND(NOT(okbuz),AND(NOT(normw),NOT(okgp))));
cj4 = AND(okgp,AND(NOT(okbuz),AND(NOT(normw),NOT(goods))));
_3bad = OR(cj1,OR(cj2,OR(cj3,cj4)));
_4bad = AND(NOT(okbuz),AND(NOT(normw),AND(NOT(goods),NOT(okgp))));
```

```
C4->scale(C4,AND(OR(smalh,modrh),OR(presx,    OR(_3bad,_4bad))));
C5->scale(C5,AND(OR(smalh,modrh),OR(NOT(neglx),OR(_2bad,_3bad))));
C6->scale(C6,AND(okhght,OR( OR(NOT(okbuz),NOT(neglx)), _2bad )));
conj = AND(okhght,AND(okbuz,AND(neglx,_2bad)));
C7->scale(C7,OR(conj,OR(runfil,AND(smalh,AND(NOT(presx),fourok)))));
C8->scale(C8,AND(okbuz,AND(okhght,AND(neglx,_1bad))));
C9->scale(C9,AND(okhght,AND(neglx,fourok)));
CONCLUSION->cj( CONCLUSION, C1, DISJ );
CONCLUSION->cj( CONCLUSION, C2, DISJ );
CONCLUSION->cj( CONCLUSION, C3, DISJ );
CONCLUSION->cj( CONCLUSION, C4, DISJ );
CONCLUSION->cj( CONCLUSION, C5, DISJ );
CONCLUSION->cj( CONCLUSION, C6, DISJ );
CONCLUSION->cj( CONCLUSION, C7, DISJ );
CONCLUSION->cj( CONCLUSION, C8, DISJ );
CONCLUSION->cj( CONCLUSION, C9, DISJ );
double concl;
(void)CONCLUSION->fcentroid( CONCLUSION, &concl );
bandStats_.qual(idx,float(concl));
DestructCFuzzySet(CONCLUSION);
DestructCFuzzySet(C1);
DestructCFuzzySet(C2);
DestructCFuzzySet(C3);
DestructCFuzzySet(C4);
DestructCFuzzySet(C5);
DestructCFuzzySet(C6);
DestructCFuzzySet(C7);
DestructCFuzzySet(C8);
DestructCFuzzySet(C9);
}
DestructCFuzzySet(tinyH);
```

```
 DestructCFuzzySet(smalH);
 DestructCFuzzySet(modrH);
 DestructCFuzzySet(normH);
 DestructCFuzzySet(tallH);
 DestructCFuzzySet(presX);
 DestructCFuzzySet(neglX);
 DestructCFuzzySet(normW);
 DestructCFuzzySet(goodS);
 DestructCFuzzySet(goodG);
 DestructCFuzzySet(goodB);
}

/*****************************************************************
 * FILE:        blindeconv.cxx
 * AUTHOR:  Andy Marks
*/
#include <basecall/mb.hxx>
#include <nrc/Complex.hxx>
#if defined(DEBUG)
static void
dzp( double const* p, int n, char const* fname )
{
  FILE* fp = ::fopen(fname,"w");
  if(NULL != fp) {
    Complex const* z = (Complex const*)&p[-1];
    for(int idx=1; idx<=n; idx++)
      ::fprintf(fp,"%11.6f %11.6f\n",z[idx].real(),z[idx].imag());
    ::fclose(fp);
    }
}
static void
```

```
drp( double const* p, int n, char const* fname )
{
  FILE* fp = ::fopen(fname,"w");
  if(NULL != fp) {
   for(int idx=1; idx<=n; idx++)
    ::fprintf(fp,"%11.6f\n",p[idx]);
   ::fclose(fp);
   }
}
#endif
static void
fftshift( double* v, int n )
{
  int idx, mid = n/2;
  for(idx=1; idx<=mid; idx++) {
   double t = v[idx];
   v[idx] = v[mid+idx];
   v[mid+idx] = t;
   }
}
static void
dCadd( double* c1, double const* c2, int n )
{
  Complex* z1 = (Complex*)&c1[-1];
  Complex const* z2 = (Complex const*)&c2[-1];
  for(int idx=1; idx<=n; idx++)
   z1[idx] += z2[idx];
}
static void
dCln( double* pc, int n )
{
```

```
    Complex* z = (Complex*)&pc[-1];
    for(int idx=1; idx<=n; idx++)
     z[idx].ln();
}
static void
dCexp( double* pc, int n )
{
    Complex* z = (Complex*)&pc[-1];
    for(int idx=1; idx<=n; idx++)
     z[idx].exp();
}
static void
dRCmul( double* pc, double const* pr, int n )
{
    Complex* z = (Complex*)&pc[-1];
    for(int idx=1; idx<=n; idx++)
     z[idx].cmul(pr[idx]);
}
static void
dRkCmul( double* pc, double Rk, int n )
{
    Complex* z = (Complex*)&pc[-1];
    for(int idx=1; idx<=n; idx++)
     z[idx].cmul( Rk );
}
enum RI { DREAL, DIMAG };
static void
dGetComponent( double const* pin, double* pout, RI ri, int n )
{
 int idx, kdx;
 if(DREAL == ri) { idx = 1; kdx = 1; }
```

```
    else        { idx = 2; kdx = -1; }
    for( ; idx<=2*n; idx+=2) {
      pout[idx+kdx] = 0.0;
      pout[idx] = pin[idx];
    }
}
static void
dCNozeros( double* ivec, int n )
{
    Complex* z = (Complex*)&ivec[-1];
    for(int idx=1; idx<=n; idx++)
      if(z[idx] == 0.0)
        z[idx].real( DBL_EPSILON );
}
void
SegRead::blindeconv( Wvfm& wv, int FBW )
{
#if defined(DEBUG)
    static char call = 0;
#endif
    int lane, jdx, sdx;
#if defined(DEBUG)
    call++;
#endif
    if(FBW != pmb_->lastFbw_) {
      double fsigma, alpha, c;
      pmb_->lastFbw_ = FBW;
      fsigma = double((FBW-1)*2)*NR_PI/double(NPTS);
      alpha  = 0.5*fsigma*fsigma;
      c = ::sqrt(NR_PI/alpha);
      for(sdx=1; sdx<=NPTS; sdx++)
```

```
        pmb_->filter_[sdx] = c * ::exp( -pmb_->ww_[sdx] / (4*alpha) );
       ::fftshift( pmb_->filter_, NPTS );
       }
      for(lane=1; lane<=4; lane++) {
       for(jdx=sdx=1; sdx<=NPTS; jdx+=2, sdx+=1) {
        pmb_->ivec_[jdx] = wv.sc_la( sdx, lane );
        pmb_->ivec_[jdx+1] = 0.0;
        }
       ::fftshift( pmb_->ivec_, 2*NPTS );
       ::dfour1( pmb_->ivec_, NPTS, 1 );
       ::dCNozeros( pmb_->ivec_, NPTS );
       ::dCln( pmb_->ivec_, NPTS );
       ::dGetComponent(pmb_->ivec_,pmb_->imag_,DIMAG,NPTS);
          ::dGetComponent( pmb_->ivec_, pmb_->ivec_, DREAL, NPTS );
       ::dfour1( pmb_->ivec_, NPTS, -1 );
       ::dRkCmul( pmb_->ivec_, 1.0/double(NPTS), NPTS );
    #if defined(DEBUG)
       if(1==call) drp(pmb_->lifter_,NPTS,"lifter");
       if(1==call) dzp(pmb_->ivec_,NPTS,"cepstrum");
    #endif
       ::dRCmul( pmb_->ivec_, pmb_->lifter_, NPTS );
       ::dfour1( pmb_->ivec_, NPTS, 1 );
       ::dGetComponent(pmb_->ivec_,pmb_->ivec_,DREAL,NPTS );
       ::dCadd( pmb_->ivec_, pmb_->imag_, NPTS );
       ::dCexp( pmb_->ivec_, NPTS );
       ::dRCmul( pmb_->ivec_, pmb_->filter_, NPTS );
       ::dfour1( pmb_->ivec_, NPTS, -1 );
       ::dRkCmul( pmb_->ivec_, 1.0/double(NPTS), NPTS );
       ::fftshift( pmb_->ivec_, 2*NPTS );
       if(1==pmb_->remune())
        for(sdx=jdx=1; sdx <= NPTS; sdx+=1, jdx+=2)
```

```
        wv.sc_la_set(sdx.lane, pmb_->ivec_[jdx]);
      }
    }

/*****************************************************************************
 * FILE:        Centroid.cxx
 * AUTHOR:  Andy Marks
 */
#include <nrc/Centroid.hxx>
int
icentroid( int const* px, int const* py, int n )
{
  int rv;
  if(1==n)
    rv = px[1];
  else if(2==n)
    rv = (px[1]+px[2]+1)/2;
  else {
    long numer=0L, denom=0L;
    for(int i=2; i<=n; i++) {
      long t = py[i-1]*(2*px[i-1] + px[i]) + py[i]*(px[i-1]+2*px[i]);
      numer = numer + t*(px[i]-px[i-1]);
      denom = denom + (px[i] - px[i-1])*(py[i-1]+py[i]);
      }
    rv = int(0.5f + float(numer)/float(3L*denom));
    }
  return rv;
}
float
fcentroid( float const* px, float const* py, int n )
{
```

```
    float rv;
    if(1==n)
      rv = px[1];
    else if(2==n)
      rv = (px[1]+px[2])/2.0f;
    else {
      double numer=0.0, denom=0.0;
      for(int i=2; i<=n; i++) {
        double t = py[i-1]*(2.0*px[i-1] + px[i]) + py[i]*(px[i-1]+2.0*px[i]);
        numer = numer + t*(px[i]-px[i-1]);
        denom = denom + (px[i] - px[i-1])*(py[i-1]+py[i]);
        }
      rv = float(numer/3.0*denom);
      }
    return rv;
  }

  /*****************************************************************************
   * FILE:        Centroid.hxx
   * AUTHOR:  Andy Marks
   */
  #if !defined(_CENTROID_HXX_)
  # define _CENTROID_HXX_
  #include <nrc/nrutil.hxx>
  DLLexport int icentroid(int const* px, int const* py, int n);
  DLLexport float centroid(float const* px, float const* py, int n);
  #endif

  /*****************************************************************************
   * FILE:        Complex.cxx
   * AUTHOR:  Andy Marks
```

```
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <nrc/Complex.hxx>
Complex const&
Complex::operator=(Complex const& b)
{
  if(this != &b) {
    r_ = b.r_;
    i_ = b.i_;
    }
  return *this;
}
Complex
Complex::operator+(Complex const& b) const
{
  Complex c;
  c.r_ = r_ + b.r_;
  c.i_ = i_ + b.i_;
  return c;
}
Complex const&
Complex::operator+=(Complex const& b)
{
  *this = *this + b;
  return *this;
}
Complex
Complex::operator*(Complex const& b) const
{
  Complex c;
  c.r_ = r_*b.r_ - i_*b.i_;
```

```
  c.i_ = i_*b.r_ + r_*b.i_;
  return c;
}
Complex const&
Complex::operator*=(Complex const& b)
{
  *this = *this * b;
  return *this;
}
Complex
Complex::operator-(Complex const& b) const
{
  Complex c;
  c.r_ = r_ - b.r_;
  c.i_ = i_ - b.i_;
  return c;
}
Complex const&
Complex::operator-=(Complex const& b)
{
  *this = *this - b;
  return *this;
}
Complex
Complex::operator/(Complex const& b) const
{
  Complex c;
  double R, D;
  if(::fabs(b.r_) >= ::fabs(b.i_)) {
    if(0.0 == b.r_) return c;
    R   = b.i_ / b.r_;
```

```
    D   = b.r_ + R*b.i_;
    if(0.0 == D)   return c;
    c.r_ = (r_ + R*i_)/D;
    c.i_ = (i_ - R*r_)/D;
    }
  else {
    if(0.0 == b.i_) return c;
    R   = b.r_ / b.i_;
    D   = b.i_ + R*b.r_;
    if(0.0 == D)   return c;
    c.r_ = (R*r_ + i_)/D;
    c.i_ = (R*i_ - r_)/D;
    }
  return c;
}
Complex const&
Complex::operator/=(Complex const& b)
{
  *this = *this / b;
  return *this;
}
void
Complex::exp()
{
   double ez = ::exp( r_ );
   r_ = ez * ::cos( i_ );
   i_ = ez * ::sin( i_ );
}
void
Complex::ln()
{
```

```
    double ang = angle();
    r_ = 0.5 * ::log( r_*r_ + i_*i_ );
    i_ = ang;
  }
  void
  Complex::conj()
  {
    i_ = -i_;
  }
  void
  Complex::cmul( double k )
  {
    r_ *= k;
    i_ *= k;
  }
  void
  dCMul( double const* a, double const* b, double* c, int n)
  {
    Complex const *za = (Complex const*)&a[-1],
              *zb = (Complex const*)&b[-1];
    Complex      *zc = (Complex*)&c[-1];
    for(int idx=1;idx<=n;idx++)
     zc[idx] = za[idx] * zb[idx];
  }

  /*************************************************************************
  * FILE:        Complex.hxx
  * AUTHOR:  Andy Marks
  * COPYRIGHT (c) 1996, University of Utah
  */
  #ifndef _COMPLEX_HXX_
```

```
#define _COMPLEX_HXX_
#include <math.h>
class Complex
{
public:
  Complex( double real = 0.0, double imag = 0.0 ) : r_(real), i_(imag) {};
  Complex( double const nr[2] ) : r_(nr[0]), i_(nr[1]) {};
  Complex( Complex const& rhs ) : r_(0.0), i_(0.0) { *this = rhs; };
  ~Complex() {};
  Complex const& operator=(Complex const& b);
  Complex operator+(Complex const& b) const;
  Complex operator*(Complex const& b) const;
  Complex operator-(Complex const& b) const;
  Complex operator/(Complex const& b) const;
  int    operator==(Complex const& b) const {return(r_==b.r_ && i_==b.i_);}
  Complex const& operator+=(Complex const& b);
  Complex const& operator*=(Complex const& b);
  Complex const& operator-=(Complex const& b);
  Complex const& operator/=(Complex const& b);
  void real(double r) { r_ = r; }
  void imag(double i) { i_ = i; }
  void exp();
  void ln();
  void conj();
  void cmul(double k);
  double  real()          const { return r_; }
  double  imag()          const { return i_; }
  double  abs()           const { return ::sqrt(r_*r_ + i_*i_); }
  double  angle()         const { return ::atan2(i_,r_); }
private:
  double r_,
```

```
        i_;
};
void dCMul( double const* a, double const* b, double* c, int n);
#endif

/*****************************************************************************
 * FILE:        dfour1.cxx
 * TYPIST:      Andy Marks
 */
#include <nrc/nr.hxx>
#define SWAP(a,b) { tempr=(a); (a)=(b); (b)=tempr; }
#if !defined(SA)
void
dfour1( double data[], unsigned long nn, int isign )
{
 unsigned long n, mmax, m, j, istep, i;
 double wtemp, wr,wpr,wpi,wi,theta;
 double tempr, tempi;
 n = nn<<1;
 j=1;
 for(i=1;i<n;i+=2) {
  if(j>i) {
    SWAP(data[j],data[i]);
    SWAP(data[j+1],data[i+1]);
    }
  m = n>>1;
  while(m >= 2 && j>m) {
   j -= m;
   m >>= 1;
    }
  j+=m;
```

```
    }
  mmax = 2;
  while(n>mmax) {
    istep = mmax<<1;
    theta = isign*(2.0*NR_PI/mmax);
    wtemp = sin(0.5*theta);
    wpr = -2.0*wtemp*wtemp;
    wpi = sin(theta);
    wr = 1.0;
    wi = 0.0;
    for(m=1;m<mmax;m+=2) {
      for(i=m;i<=n;i+=istep) {
        j=i+mmax;
        tempr = wr*data[j] - wi*data[j+1];
        tempi = wr*data[j+1]+wi*data[j];
        data[j]   = data[i]-tempr;
        data[j+1]  = data[i+1]-tempi;
        data[i]   += tempr;
        data[i+1] += tempi;
        }
      wr = (wtemp=wr)*wpr-wi*wpi+wr;
      wi = wi*wpr + wtemp*wpi + wi;
      }
    mmax = istep;
    }
  }
  #endif
  #if defined(SA)
  #include <stdio.h>
  void
  main()
```

```
{
 double *unitstep;
 size_t idx, jdx;
 if(NULL == (unitstep = dvector(1,16)))
   nrerror("main: failed to allocate unitstep vector\n");
 unitstep[ 1] = 1.0; unitstep[ 2] = 0.0;
 unitstep[ 3] = 1.0; unitstep[ 4] = 0.0;
 unitstep[ 5] = 0.0; unitstep[ 6] = 0.0;
 unitstep[ 7] = 0.0; unitstep[ 8] = 0.0;
 unitstep[ 9] = 0.5; unitstep[10] = 0.0;
 unitstep[11] = 0.5; unitstep[12] = 0.0;
 unitstep[13] = 0.0; unitstep[14] = 0.0;
 unitstep[15] = 0.0; unitstep[16] = 0.0;
 ::printf("INPUT:\n");
 for(idx = 1; idx<=16; idx++)
   printf("\tv[%2ld]=%11.6f\n",idx,unitstep[idx]);
 dfour1( unitstep, 8, 1 );
 ::printf("FFT(INPUT):\n");
 for(idx = 1; idx<=16; idx++)
   printf("\tv[%2ld]=%11.6f\n",idx,unitstep[idx]);
 dfour1( unitstep, 8, -1 );
 ::printf("IFFT(INPUT):\n");
 for(idx = 1; idx<=16; idx++)
   printf("\tv[%2ld]=%11.6f\n",idx,unitstep[idx]);
 free_dvector(unitstep,1,16);
}
#endif

/*************************************************************************
 * FILE: fgapcheck.c
 * COPYRIGHT (c) 1996, University of Utah
```

```
 */
#include <basecall/fuzzyset.h>
#include <basecall/fgapcheck.h>
#if defined(SA)
# define VERBOSE
#else
# undef VERBOSE
#endif
static double
weight(size_t n, size_t gs, size_t cs)
{
 return 1.0-sqrt((double)((n-gs)*(n-gs) + (n-cs)*(n-cs))/(2.0*(double)(n*n)));
}
static double
gcness(size_t n, size_t gs, size_t cs, size_t mx, size_t my)
{
 size_t lh, sh;
 double normalize, rv;
 lh = (n+1)/2; sh = n-lh;
 if(0==mx) mx = lh; if(0==my) my = sh;
 normalize = weight(n,mx,my);
 rv = weight(n,gs,cs)/normalize;
 if(rv>1.0) rv = 1.0;
 return rv*rv;
}
int
gapcheck(
 int const* pis,
 int const* piw
 int const* ps,
 int const* pw,
```

```
    char const* pseq,
    size_t nbands,
    float** output
    )
{
# define NPREV 5
  static double const bigwidX[] = {0.0, 1.0};
  static double const bigwidY[] = {0.0, 1.0};
# define BGWDSZ (sizeof(bigwidY)/sizeof(bigwidY[0]))
  static double const biggapX[] = {0.42,0.6,1.0};
  static double const biggapY[] = {0.0,1.0,1.0};
# define BGGPSZ (sizeof(biggapY)/sizeof(biggapY[0]))
  static double const mdgpX[] = { 0.25, 0.3, 0.45, 0.57 };
  static double const mdgpY[] = { 0.0,  1.0, 1.0,  0.0 };
# define MDGPSZ (sizeof(mdgpY)/sizeof(mdgpY[0]))
  static double const smgpX[] = {-1.0, -0.5, 0.0};
  static double const smgpY[] = { 1.0,  1.0, 0.0};
# define SMGPSZ (sizeof(smgpY)/sizeof(smgpY[0]))
# define A 0.5079
# define B 1.5002
# define C 1.5069
# define D 2.5063
  static double const norm_x[] = { A,  B,  C };
  static double const norm_y[] = { 1.0, 1.0, 0.0 };
# define NORMSZ (sizeof(norm_x)/sizeof(norm_x[0]))
  static double const split_x[] = { B,  C,  D };
  static double const split_y[] = { 0.0, 1.0, 1.0 };
# define SPLITSZ (sizeof(split_x)/sizeof(split_x[0]))
  static double const concl_x[] = { A,  D };
  static double const concl_y[] = { 0.0, 0.0 };
# define CONCLSZ (sizeof(concl_x)/sizeof(concl_x[0]))
```

```
size_t *gs,
    *cs;
double *gc;
CFuzzySet *ciBgGP, *ciMdGP, *ciSmGP, *ciBgWD;
size_t idx;
int rv = 1;
gc = (double*)malloc(sizeof(*gc)*(nbands+1));
gs = (size_t*)malloc(sizeof(*gs)*(nbands+1));
cs = (size_t*)malloc(sizeof(*cs)*(nbands+1));
if(NULL==gs || NULL==cs || NULL==gc) {
 fprintf(stderr,"out of memory in fgapcheck.c, line %d\n",__LINE__);
 if(NULL != gs) { free(gs); gs = NULL; }
 if(NULL != cs) { free(cs); cs = NULL; }
 if(NULL != gc) { free(gc); gc = NULL; }
 return 0;
 }
ciBgWD = ConstructCFuzzySet( BGWDSZ, bigwidX, bigwidY, CONTINT ),
ciBgGP = ConstructCFuzzySet( BGGPSZ, biggapX, biggapY, VERY );
ciMdGP = ConstructCFuzzySet( MDGPSZ, mdgpX, mdgpY, NOHEDGE );
ciSmGP = ConstructCFuzzySet( SMGPSZ, smgpX, smgpY, CONTINT );
for(idx=1; idx<=nbands; idx++) {
 size_t bgnj, jdx;
 double expgaps, prvgaps = 0.0;
 bgnj = (idx>NPREV)?(idx-NPREV):1;
 gs[idx] = cs[idx] = 0;
 for(jdx = bgnj; jdx < idx; jdx++) {
  prvgaps += (double)ps[jdx];
  switch( pseq[jdx] ) {
   case 'G': gs[idx] += 1; break;
   case 'C': cs[idx] += 1; break;
   default:  break;
```

```
    }
   }
  expgaps = (double)(piw[idx]*(idx-bgnj));
  if((0.0==expgaps) || (prvgaps<expgaps))
   prvgaps = expgaps = 1.0;
  gc[idx] = (expgaps/prvgaps)*gcness(NPREV,gs[idx],cs[idx],NPREV/2,NPREV/2);
  }
 for(idx=1; idx<=nbands; idx++) {
  CFuzzySet *CONCLUSION, *R_SPLIT, *R_NORM;
# define N 1
  double rawgap[N+1], rawwid[N+1], biggap[N+1], smlgap[N+1], bigwid[N+1], gcrich;
  double conj1, conj2, cmpti, concl;
  double medgap[N+1], fatso;
  R_NORM  = ConstructCFuzzySet( NORMSZ,  norm_x,  norm_y,  NOHEDGE );
  R_SPLIT = ConstructCFuzzySet( SPLITSZ, split_x, split_y, NOHEDGE );
  CONCLUSION = ConstructCFuzzySet( CONCLSZ, concl_x, concl_y, NOHEDGE );
  if(0==pis[idx] || 0==piw[idx]) {
   DestructCFuzzySet(R_SPLIT);
   DestructCFuzzySet(R_NORM);
   DestructCFuzzySet(CONCLUSION);
   rv = 0;
   break;
   }
  rawgap[N]   = ((double)ps[idx]/(double)pis[idx]) - 1.0;
  rawgap[N-1] = (1==idx)? 0.0: (((double)ps[idx-1]/(double)pis[idx]) - 1.0);
  rawwid[N]   = ((double)pw[idx]/(double)piw[idx]) - 1.0;
  rawwid[N-1] = (1==idx)? 0.0: (((double)pw[idx-1]/(double)piw[idx]) - 1.0);
  biggap[N]   = ciBgGP->membership( ciBgGP, rawgap[N] );
  medgap[N]   = ciMdGP->membership( ciMdGP, rawgap[N] );
  smlgap[N]   = ciSmGP->membership( ciSmGP, rawgap[N] );
  biggap[N-1] = ciBgGP->membership( ciBgGP, rawgap[N-1] );
```

```
    medgap[N-1] = ciMdGP->membership( ciMdGP, rawgap[N-1] );
    smlgap[N-1] = ciSmGP->membership( ciSmGP, rawgap[N-1] );
    bigwid[N]   = ciBgWD->membership( ciBgWD, rawwid[N] );
    bigwid[N-1] = ciBgWD->membership( ciBgWD, rawwid[N-1] );
    gcrich = gc[idx];
    fatso = AND(bigwid[N-1],AND(medgap[N-1],medgap[N]));
#if defined(VERBOSE)
    printf("  pseq[%3d]=%c      pseq[%3d]=%c\n",idx-1,pseq[idx-1],idx,pseq[idx]);
    printf("    ps[%3d]=%2d       ps[%3d]=%d\n",idx-1,ps[idx-1],  idx,ps[idx]);
    printf("   pis[%3d]=%2d      pis[%3d]=%d\n",idx-1,pis[idx-1], idx,pis[idx]);
    printf("    pw[%3d]=%2d       pw[%3d]=%d\n",idx-1,pw[idx-1],  idx,pw[idx]);
    printf("   piw[%3d]=%2d      piw[%3d]=%d\n",idx-1,piw[idx-1], idx,piw[idx]);
    printf("rawgap[%3d]=%5.2f rawgap[%3d]=%5.2f\n",idx-1,rawgap[N-1],idx,rawgap[N]);
    printf("rawwid[%3d]=%5.2f rawwid[%3d]=%5.2f\n",idx-1,rawwid[N-1],idx,rawwid[N]);
    printf("biggap[%3d]=%5.2f biggap[%3d]=%5.2f\n",idx-1,biggap[N-1],idx,biggap[N]);
    printf("medgap[%3d]=%5.2f medgap[%3d]=%5.2f\n",idx-1,medgap[N-1],idx,medgap[N]);
    printf("smlgap[%3d]=%5.2f smlgap[%3d]=%5.2f\n",idx-1,smlgap[N-1],idx,smlgap[N]);
    printf("bigwid[%3d]=%5.2f bigwid[%3d]=%5.2f\n",idx-1,bigwid[N-1],idx,bigwid[N]);
    printf("gcrich = %4.2f\n",gcrich);
    printf("fatso  = %4.2f\n",fatso);
#endif
    conj1 = AND(biggap[N],gcrich);
    conj2 = AND(biggap[N],smlgap[N-1]);
    conj2 = AND(conj2,NOT(bigwid[N]) );
    conj2 = AND(conj2,NOT(bigwid[N-1]) );
    R_NORM->scale( R_NORM, OR(NOT(biggap[N]),OR(conj1,conj2)) );
    conj1 = AND(biggap[N],OR(bigwid[N-1],bigwid[N]));
    conj2 = AND(biggap[N],AND(NOT(smlgap[N-1]),NOT(gcrich)));
    R_SPLIT->scale( R_SPLIT, OR(conj1,conj2));
    CONCLUSION->cj( CONCLUSION, R_NORM,  DISJ );
    CONCLUSION->cj( CONCLUSION, R_SPLIT, DISJ );
```

```
    (void)CONCLUSION->fcentroid( CONCLUSION, &concl );
    cmpti = CONCLUSION->compatIndex( CONCLUSION );
    output[idx][1] = (float)concl;
    output[idx][2] = (float)cmpti;
#if defined(VERBOSE)
    R_NORM->print( R_NORM, "R_NORM" );
    R_SPLIT->print( R_SPLIT, "R_SPLIT" );
    CONCLUSION->print( CONCLUSION, "CONCLUSION" );
    printf("idx=%d  concl=%f  cmpti=%f\n\n".idx,concl,cmpti);
#endif
    DestructCFuzzySet(R_SPLIT);
    DestructCFuzzySet(R_NORM);
    DestructCFuzzySet(CONCLUSION);
    }
  DestructCFuzzySet(ciBgWD);
  DestructCFuzzySet(ciBgGP);
  DestructCFuzzySet(ciSmGP);
  DestructCFuzzySet(ciMdGP);
  free(gc);
  free(gs); free(cs);
  return rv;
}
#if defined(SA)
#include <stdio.h>
#include <nrc/nrutil.hxx>
int
main(int argc, char* argv[])
{
    static int sp[6], wd[6], isp[6], iwd[6], idx;
    char *p, *phd;
    if(2!=argc) {
```

```
fprintf(stderr,"usage: %s argstr\n",argv[0]);
fprintf(stderr," where: argstr is a [:] delimited set of 7 fields\n");
fprintf(stderr," fld1: sp(n-1)\n");
fprintf(stderr," fld2: sp(n)\n");
fprintf(stderr," fld3: gp(n-1)\n");
fprintf(stderr," fld4: gp(n)\n");
fprintf(stderr," fld5: Exp[sp]\n");
fprintf(stderr," fld6: Exp[gp]\n");
fprintf(stderr," fld7: seq[n-5..n]\n");
fprintf(stderr," Ex: %s 9:8:8:9:10:10:GCGCG\n",argv[0]);
return 1;
}
for(idx=0;idx<6;idx++)
 sp[idx] = wd[idx] = isp[idx] = iwd[idx] = 10;
if(NULL != (p = strchr(phd=argv[1],':'))) {
*p++ = '\0'; sp[4] = atoi(phd); phd = p;
if(NULL != (p = strchr(phd,':'))) {
*p++ = '\0'; sp[5] = atoi(phd); phd = p;
if(NULL != (p = strchr(phd,':'))) {
*p++ = '\0'; wd[4] = atoi(phd); phd = p;
if(NULL != (p = strchr(phd,':'))) {
*p++ = '\0'; wd[5] = atoi(phd); phd = p;
if(NULL != (p = strchr(phd,':'))) {
*p++ = '\0'; isp[5] = atoi(phd); phd = p;
if(NULL != (p = strchr(phd,':'))) {
float** output = matrix(1L,5L,1L,2L);
*p++ = '\0'; iwd[5] = atoi(phd); phd = p;
printf("%d %d %d %d %d %s\n",
 sp[4],sp[5],wd[4],wd[5],isp[5],iwd[5],phd);
gapcheck( isp, iwd, sp, wd, phd, 5, output );
printf("%f %f\n",output[5][1],output[5][2]);
```

```
            return 0;
            }
           }
          }
         }
        }
       }
       fprintf(stderr,"%s: missing input field(s): [%s]\n",argv[0],phd);
       return 1;
}
#endif

/*****************************************************************************
 * FILE: fomitokn.c
 * COMP: c89 -DSA omitnok.c fuzzyset.o -o omitokn -lm
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <float.h>
#include <basecall/fuzzyset.h>
int
omitokn(
 int const* pinS, float const* pht, float const* plo, int const* pLsp,
 int const* pRsp, float const* pxb, int NPK, float** output )
{
 static double const okSpX[] = {0.2, 0.5, 0.8};
 static double const okSpY[] = {1.0, 0.0, 1.0};
#if defined(AS_WAS)
 static double const abSpX[] = {0.3, 0.4, 0.6, 0.7};
#else
 static double const abSpX[] = {0.2, 0.4, 0.6, 0.8};
#endif
```

```
 static double const abSpY[] = {0.0, 1.0, 1.0, 0.0};
# define OKSPSZ (sizeof(okSpY)/sizeof(okSpY[0]))
# define ABSPSZ (sizeof(abSpY)/sizeof(abSpY[0]))
 static double const tiXbX[] = {1.0, 1.4, 1.8};
 static double const tiXbY[] = {1.0, 0.5, 0.0};
 static double const okXbX[] = {1.2, 1.4};
 static double const okXbY[] = {0.0, 1.0};
# define TIXbSZ (sizeof(tiXbX)/sizeof(tiXbX[0]))
# define OKXbSZ (sizeof(okXbX)/sizeof(okXbX[0]))
 static double tiHtX[] = {0.02,0.07};
 static double const tiHtY[] = {1.00,0.00};
 static double okHtX[] = {0.01,0.06};
 static double const okHtY[] = {0.00,1.00};
# define TIHTSZ (sizeof(tiHtX)/sizeof(tiHtX[0]))
# define OKHTSZ (sizeof(okHtX)/sizeof(okHtX[0]))
 static double const ok_x[] = {0.4596, 1.3797, 1.6864};
 static double const ok_y[] = {1.0000, 1.0000, 0.0000};
# define OK_SZ (sizeof(ok_x)/sizeof(ok_x[0]))
 static double const n_x[] = {1.3797, 1.6864, 2.2998, 2.6065};
 static double const n_y[] = {0.0000, 1.0000, 1.0000, 0.0000};
# define N_SZ (sizeof(n_x)/sizeof(n_x[0]))
 static double const omit_x[] = {2.2998, 2.6065, 3.5540};
 static double const omit_y[] = {0.0000, 1.0000, 1.0000};
# define OMIT_SZ (sizeof(omit_x)/sizeof(omit_x[0]))
 static double  const concl_x[] = {0.4596, 3.5540};
 static double  const concl_y[] = {0.0000, 0.0000};
# define CONCL_SZ (sizeof(concl_x)/sizeof(concl_x[0]))
 CFuzzySet *okSP, *abSP, *tiXb, *okXb, *tiHT, *okHT;
 double mean_plo = 0.0;
 int idx, rv=1;
 for(idx=1:idx<=NPK:idx++) mean_plo += plo[idx];
```

```
mean_plo /= (double)NPK;
tiHtX[0] = 0.4*mean_plo; tiHtX[1] = 1.1*mean_plo;
okHtX[0] = 0.5*mean_plo; okHtX[1] = 1.5*mean_plo;
okSP = ConstructCFuzzySet( OKSPSZ, okSpX, okSpY, NOHEDGE );
#if defined(AS_WAS)
abSP = ConstructCFuzzySet( ABSPSZ, abSpX, abSpY, NOHEDGE );
#else
abSP = ConstructCFuzzySet( ABSPSZ, abSpX, abSpY, VERY );
#endif
tiXb = ConstructCFuzzySet( TIXbSZ, tiXbX, tiXbY, NOHEDGE );
okXb = ConstructCFuzzySet( OKXbSZ, okXbX, okXbY, NOHEDGE );
tiHT = ConstructCFuzzySet( TIHTSZ, tiHtX, tiHtY, NOHEDGE );
okHT = ConstructCFuzzySet( OKHTSZ, okHtX, okHtY, SOMEWHAT );
for(idx=1; idx<=NPK; idx++) {
 CFuzzySet *CONCLUSION,*RULE_OK, *RULE_N, *RULE_OMIT;
 double okht,tiht,
     okLsp,okRsp,oksp,
     abLsp,abRsp,absp,
     tixb,okxb;
 double concl,cmpti, modLsp, modRsp, insp = pinS[idx];
 CONCLUSION = ConstructCFuzzySet( CONCL_SZ, concl_x, concl_y, NOHEDGE );
 RULE_OK = ConstructCFuzzySet( OK_SZ, ok_x, ok_y, NOHEDGE );
 RULE_N = ConstructCFuzzySet( N_SZ, n_x, n_y, NOHEDGE );
 RULE_OMIT = ConstructCFuzzySet( OMIT_SZ, omit_x, omit_y, NOHEDGE );
 if(pLsp[idx] < (insp/2.0)) modLsp = insp/2.0;
 else                modLsp = fmod( pLsp[idx], insp );
 if(pRsp[idx] < (insp/2.0)) modRsp = insp/2.0;
 else                modRsp = fmod( pRsp[idx], insp );
 if(0.0 == insp) {
  DestructCFuzzySet(CONCLUSION);
  DestructCFuzzySet(RULE_OK);
```

```
        DestructCFuzzySet(RULE_N);
        DestructCFuzzySet(RULE_OMIT);
        rv = 0;
        break;
        }
      modLsp /= insp;
      modRsp /= insp;
      okLsp  = okSP->membership( okSP, modLsp );
      okRsp  = okSP->membership( okSP, modRsp );
      abLsp  = abSP->membership( abSP, modLsp );
      abRsp  = abSP->membership( abSP, modRsp );
      oksp   = OR(okLsp,okRsp);
      absp   = OR(abLsp,abRsp);
      tixb   = tiXb->membership( tiXb, pxb[idx] );
      okxb   = okXb->membership( okXb, pxb[idx] );
      tiht   = tiHT->membership( tiHT, pht[idx] );
      okht   = okHT->membership( okHT, pht[idx] );
    #if defined(AS_WAS)
      RULE_OK->scale( RULE_OK, AND(okxb,okht));
    #else
      RULE_OK->scale( RULE_OK, AND(okxb,OR(okht,oksp)) );
    #endif
      RULE_N->scale( RULE_N, AND(tixb,OR(okht,AND(oksp,tiht))));
      RULE_OMIT->scale( RULE_OMIT, AND(tiht,absp) );
      CONCLUSION->cj( CONCLUSION, RULE_OK, DISJ );
      CONCLUSION->cj( CONCLUSION, RULE_N, DISJ );
      CONCLUSION->cj( CONCLUSION, RULE_OMIT, DISJ );
      (void)CONCLUSION->fcentroid( CONCLUSION, &concl );
      cmpti = CONCLUSION->compatIndex( CONCLUSION );
      output[idx][1] = (float)concl;
      output[idx][2] = (float)cmpti;
```

```
        DestructCFuzzySet(CONCLUSION);
        DestructCFuzzySet(RULE_OK);
        DestructCFuzzySet(RULE_N);
        DestructCFuzzySet(RULE_OMIT);
    }
    DestructCFuzzySet(okSP);
    DestructCFuzzySet(abSP);
    DestructCFuzzySet(okXb);
    DestructCFuzzySet(tiXb);
    DestructCFuzzySet(okHT);
    DestructCFuzzySet(tiHT);
    return rv;
}
#if defined(SA)
void
main(int argc, char* argv[])
{
#if 1
# define HTINC 0.01
# define SPINC 1.0
# define XbINC 0.2
#else
# define HTINC 0.1
# define SPINC 1.0
# define XbINC 0.5
#endif
# define OK_IDX   0
# define N_IDX    1
# define OMIT_IDX 2
    double normSP = 12.0;
    double ht, sp, xb, n[4];
```

```
n[0] = n[1] = n[2] = n[3] = 0.0;
printf("Sweeping [ht] from 0.0 to 0.2 in steps of %4.2f\n",HTINC);
printf("Sweeping [sp] from 0.0 to %4.1f in steps of %4.2f\n",normSP,SPINC);
printf("Sweeping [xb] from 1.0 to 2.0 in steps of %4.2f\n",XbINC);
for(ht = 0.0;  ht <= 0.2;  ht += HTINC) {
 for(sp = 0.0; sp <= normSP; sp += SPINC)
  for(xb = 1.0; xb <= 2.0;  xb += XbINC) {
   double out[3], ci, dfz;
   char const* CALL;
   ftn_omitokn( &normSP, 1,&ht,&sp,&sp, &xb, out,&ci, &dfz);
   printf("\n{ht,sp,xb}:{%5.2f,%5.2f,%5.2f} -> ", ht,sp,xb);
   printf("{OK,N,OMIT}:{%3.2f,%3.2f,%3.2f}",
    out[OK_IDX],out[N_IDX],out[OMIT_IDX]);
   if(dfz <= 0.35) {
    CALL = "  OK"; n[0] += 1.0;
    }
   else if(dfz >= 0.65) {
    CALL = "OMIT"; n[1] += 1.0;
    }
   else {
    CALL = "   N"; n[2] += 1.0;
    }
   n[3] += 1.0;
   printf(" %s (dfz=%4.3f,%3.2f)",CALL,dfz,ci);
   }
  printf("\n");
  }
 printf("%3.2f%% OK, %3.2f%% N, %3.2f%% OMIT in %5.0f tests\n",
  n[0]/n[3], n[2]/n[3], n[1]/n[3], n[3]);
}
#endif
```

```
#if defined(USE_WSC)
# include <ab/wsc.h>
#else
# include <malloc.h>
#endif
#include <basecall/fuzzyset.h>
static double
membership( struct CFuzzySet const* pcfz, double pt ) {
 double yy;
 if(0 == pcfz->n)
  yy = 0.0;
 else if(pt <= pcfz->x[0])
  yy = pcfz->y[0];
 else if(pt >= pcfz->x[pcfz->n-1])
  yy = pcfz->y[pcfz->n-1];
 else {
  double yh, yhm1;
  int lo, mid, hi;
  lo = 0;
  hi = pcfz->n-1;
  for(;;) {
   mid = (lo+hi)/2;
   if(mid == lo)
    break;
   else if(pcfz->x[mid] < pt)
    lo = mid;
   else
    hi = mid;
   }
  yh = pcfz->y[hi];
  yhm1 = pcfz->y[hi-1];
```

```
    yy = yhm1 + (pt - pcfz->x[hi-1])/(pcfz->x[hi] - pcfz->x[hi-1])*(yh - yhm1);
    }
  return pcfz->d_phedge( yy );
}
static double
compatIndex( struct CFuzzySet const* pcfz ) {
  double ci = 0.0;
  int idx;
  for(idx = 0; idx < pcfz->n; idx++)
    if(pcfz->y[idx] > ci) ci = pcfz->y[idx];
  return pcfz->d_phedge( ci );
}
static int
invalid(struct CFuzzySet const* pcfz ) {
  return !pcfz->n;
}
static void
print(struct CFuzzySet const* pcfz, char const* pname) {
  int idx;
  printf("\nCFuzzySet:[%s]\n    x       y",pname);
  for(idx = 0; idx < pcfz->n; idx++)
    printf("\n  %8.3lf %8.3lf",pcfz->x[idx],pcfz->d_phedge(pcfz->y[idx]));
  printf("\n");
}
static void
negate( struct CFuzzySet* pcfz ) {
  int idx;
  for(idx = 0; idx < pcfz->n; idx++)
    pcfz->y[idx] = 1.0 - pcfz->y[idx];
}
static void
```

```
scale( struct CFuzzySet* pcfz, double fac ) {
 int idx;
 for(idx = 0; idx < pcfz->n; idx++)
  pcfz->y[idx] *= fac;
}
static void
cj_out( double x,double y, double* xn, double* yn, int* nn ) {
 double d;
 if((*nn > 0) &&
   (fabs(x-xn[*nn-1]) < 1.e-20) && (fabs(y-yn[*nn-1]) < 1.e20))
  return;
 if(*nn > 1) {
  int nm1 = *nn-1,
     nm2 = *nn-2;
  d = xn[nm1]*(y-yn[nm2]) + yn[nm1]*(xn[nm2]-x) + x*yn[nm2] - y*xn[nm2];
  if(fabs(d) < 1.e-10)
   --*nn;
  }
 xn[*nn] = x;
 yn[*nn] = y;
 ++*nn;
}
static void
intsec( double x1,double y1, double x2,double y2, double y3,double y4,
      double *xint, double *yint) {
 double den;
 den = y1 - y2 - y3 + y4;
 *xint = (x2*y1 - x1*y2 - x2*y3 + x1*y4) / den;
 *yint = (y1*y4 - y2*y3) / den;
}
static void
```

```
cj(struct CFuzzySet* pcfz,struct CFuzzySet const* s, LOGICAL_OPERATOR cjdj) {
 int nn. vertex0, vertex1, use_func_0;
 double *xn, *yn, xl, xr, y0l, y0r, y1l, y1r, xint, yint;
 double next_x0,next_y0,next_x1,next_y1,rightmost_x,frac;
 double (*h1)(double y), (*h2)(double y);
 int idx;
 h1 = pcfz->d_phedge;
 h2 = s->d_phedge;
 if(0 == pcfz->n)
  return;
 if(0 == s->n) {
  if(NULL != pcfz->x) {
#if defined(USE_WSC)
    FreeMemory( pcfz->x );
#else
    free( pcfz->x );
#endif
    pcfz->x = NULL;
    }
  pcfz->n = 0;
  return;
  }
#if defined(USE_WSC)
 xn = NULL;
 (void)fNewMemory((void**)&xn,sizeof(double)*4*(pcfz->n+s->n));
#else
 xn = (double*)malloc(sizeof(double)*4*(pcfz->n+s->n));
#endif
 if(NULL == xn) {
  pcfz->n = 0;
  if(NULL != pcfz->x) {
```

```
#if defined(USE_WSC)
    FreeMemory( pcfz->x );
#else
    free( pcfz->x );
#endif
    pcfz->x = NULL;
    }
  return;
  }
 yn = &xn[ 2*(pcfz->n+s->n) ];
 cjdj = (DISJ != cjdj)? CONJ: DISJ;
 if(pcfz->x[0] < s->x[0]) {
  xl = pcfz->x[0];
  vertex0 = 1;
  vertex1 = 0;
  }
 else if(pcfz->x[0] > s->x[0]) {
  xl = s->x[0];
  vertex0 = 0;
  vertex1 = 1;
  }
 else {
  xl = pcfz->x[0];
  vertex0 = vertex1 = 1;
  }
 y0l = h1(pcfz->y[0]);
 y1l = h2(s->y[0]);
 nn = 0;
 cj_out( xl, (cjdj^(y0l>y1l))?y0l:y1l, xn, yn, &nn );
 if(pcfz->x[pcfz->n-1] >= s->x[s->n-1])
  rightmost_x = pcfz->x[pcfz->n-1];
```

```
else
 rightmost_x = s->x[s->n-1];
if(vertex0 < pcfz->n) {
 next_x0 = pcfz->x[vertex0];
 next_y0 = h1(pcfz->y[vertex0]);
 }
else {
 next_x0 = rightmost_x;
 next_y0 = h1(pcfz->y[vertex0-1]);
 }
if(vertex1 < s->n) {
 next_x1 = s->x[vertex1];
 next_y1 = h2(s->y[vertex1]);
 }
else {
 next_x1 = rightmost_x;
 next_y1 = h2(s->y[vertex1-1]);
 }
while((vertex0 < pcfz->n) || (vertex1 < s->n)) {
 if(next_x0 < next_x1)
  use_func_0 = 1;
 else if(next_x1 < next_x0)
  use_func_0 = 0;
 else
  use_func_0 = (vertex0 < pcfz->n);
 if(1 == use_func_0) {
  xr = next_x0;
  y0r = next_y0;
  if(next_x1 == xl)
   frac = 0.0;
  else
```

```
        frac = (xr-xl)/(next_x1-xl);
      y1r = y1l + frac*(next_y1-y1l);
      if(++vertex0 < pcfz->n) {
       next_x0 = pcfz->x[vertex0];
       next_y0 = h1(pcfz->y[vertex0]);
       }
      else
       next_x0 = rightmost_x;
      }
    else {
      xr = next_x1;
      y1r = next_y1;
      if(next_x0 == xl)
       frac = 0.0;
      else
       frac = (xr-xl)/(next_x0 - xl);
      y0r = y0l + frac*(next_y0-y0l);
      if(++vertex1 < s->n) {
       next_x1 = s->x[vertex1];
       next_y1 = h2(s->y[vertex1]);
       }
      else
       next_x1 = rightmost_x;
      }
    if((xr>xl) && ((y0l-y1l)*(y0r-y1r) < 0.0)) {
      intsec( xl, y0l, xr, y0r, y1l, y1r, &xint, &yint );
      cj_out( xint, yint, xn, yn, &nn);
      }
    cj_out( xr, (cjdj ^ (y0r>y1r))? y0r: y1r, xn, yn, &nn );
    xl = xr;
    y0l = y0r;
```

```
    yll = ylr;
    }
  if(NULL != pcfz->x) {
#if defined(USE_WSC)
    FreeMemory( pcfz->x );
#else
    free( pcfz->x );
#endif
    pcfz->x = NULL;
    }
#if defined(USE_WSC)
  pcfz->x = NULL;
  (void)fNewMemory((void**)&pcfz->x,sizeof(double)*2*nn);
#else
  pcfz->x = (double*)malloc(sizeof(double)*2*nn);
#endif
  pcfz->y = pcfz->x + nn;
  pcfz->n = nn;
  memcpy( pcfz->x, xn, pcfz->n*sizeof(double) );
  for(idx = 0; idx < nn; idx++)
    pcfz->y[idx] = pcfz->d_punhedge( yn[idx] );
  if(NULL != xn) {
#if defined(USE_WSC)
    FreeMemory( xn );
#else
    free( xn );
#endif
    xn = NULL;
    }
  }
static int
```

```
fcentroid( struct CFuzzySet const* pcfz, double* pcentroid) {
 int rv = 0;
 *pcentroid = 0.0;
 if(pcfz->n >= 2) {
  double numer = 0.0,
       denom = 0.0;
  int idx;
  for(idx = 1; idx < pcfz->n; idx++) {
   double t, hym1, hy;
   hy = pcfz->d_phedge( pcfz->y[idx] );
   hym1 = pcfz->d_phedge( pcfz->y[idx-1] );
   t = hym1*(2.0*pcfz->x[idx-1] + pcfz->x[idx]) +
       hy*(pcfz->x[idx-1] + 2.0*pcfz->x[idx]);
   numer += t*(pcfz->x[idx] - pcfz->x[idx-1]);
   denom += (pcfz->x[idx]-pcfz->x[idx-1]) * (hym1 + hy);
   }
  if(fabs(denom) > 1.e-20) {
   *pcentroid = numer / (3.0*denom);
   rv = 1;
   }
  }
 return rv;
}
static double return_y(double yval) { return yval; }
static double square_y(double yval) { return yval*yval; }
static double sqrt_y(double yval) { return sqrt(yval); }
static double contint(double yval) {
 return (yval>=0.5)? sqrt(yval): (yval*yval);
 }
static double contdeint(double yval) {
 return (yval>=sqrt(0.5))? (yval*yval): sqrt(yval);
```

```
}
CFuzzySet*
ConstructCFuzzySet(int npts, double const* xpts, double const* ypts, HEDGE ht) {
 CFuzzySet* pcfz = NULL;
#if defined(USE_WSC)
 (void)fNewMemory((void**)&pcfz,sizeof(*pcfz));
#else
 pcfz = (CFuzzySet*)malloc(sizeof(*pcfz));
#endif
 if(NULL != pcfz) {
  if((pcfz->n = npts) < 1) {
   pcfz->n = 0;
   }
  else {
   pcfz->fcentroid = fcentroid;
   pcfz->cj = cj;
   pcfz->invalid = invalid;
   pcfz->membership = membership;
   pcfz->compatIndex = compatIndex;
   pcfz->negate = negate;
   pcfz->print = print;
   pcfz->scale = scale;
   switch(pcfz->d_ht=ht) {
    default:
     pcfz->d_ht = NOHEDGE;
    case NOHEDGE:
     pcfz->d_punhedge = return_y;
     pcfz->d_phedge = return_y;
     break;
    case VERY:
     pcfz->d_phedge = square_y;
```

```
            pcfz->d_punhedge = sqrt_y;
            break;
           case SOMEWHAT:
            pcfz->d_phedge = sqrt_y;
            pcfz->d_punhedge = square_y;
            break;
           case CONTINT:
            pcfz->d_phedge = contint;
            pcfz->d_punhedge = contdeint;
            break;
           }
      #if defined(USE_WSC)
          pcfz->x = NULL;
          fNewMemory((void**)&pcfz->x,sizeof(double)*2*npts);
      #else
          pcfz->x = (double*)malloc(sizeof(double)*2*npts);
      #endif
          if(NULL == pcfz->x) {
           pcfz->n = 0;
           }
          else {
           pcfz->y = &pcfz->x[ pcfz->n ];
           memcpy(pcfz->x,xpts,npts*sizeof(double));
           memcpy(pcfz->y,ypts,npts*sizeof(double));
           }
         }
        }
       return pcfz;
       }
      void
      DestructCFuzzySet(struct CFuzzySet* pcfz) {
```

```
#if defined(USE_WSC)
 if(NULL != pcfz) {
  if(NULL != pcfz->x) {
   FreeMemory( pcfz->x );
   pcfz->x = NULL;
   }
  FreeMemory( pcfz );
  }
#else
 if(NULL != pcfz) {
  if(NULL != pcfz->x) {
   free( pcfz->x );
   pcfz->x = NULL;
   }
  free( pcfz );
  }
#endif
 }
#if defined(SA)
static void
ftn_omitokn( double med_sp ) {
 static double t_SpX[] = {0.0, 0.25, 0.5, 0.75, 1.0};
 static double nSpY[]  = {1.0, 0.50, 0.0, 0.50, 1.0};
# define SPSZ (sizeof(nSpY)/sizeof(nSpY[0]))
 static double loSnX[] = {1.0, 1.4, 1.8};
 static double loSnY[] = {1.0, 0.5, 0.0};
# define LoSNSZ (sizeof(loSnX)/sizeof(loSnX[0]))
 static double loHtX[] = {0.03,0.05,0.07};
 static double loHtY[] = {1.00,0.50,0.00};
# define LoHTSZ (sizeof(loHtX)/sizeof(loHtX[0]))
 static double okxy[] = {0.0, 1.0};
```

```
CFuzzySet *vNSP, *swNSP, *vLoSN, *swLoSN, *vLoHT, *swLoHT;
double ht, Lsp,Rsp, sn, nSpX[ SPSZ ];
int idx;
for(idx = 0; idx < SPSZ; idx++)
  nSpX[idx] = t_SpX[idx]*med_sp;
vNSP  = ConstructCFuzzySet( SPSZ, nSpX, nSpY, VERY );
swNSP  = ConstructCFuzzySet( SPSZ, nSpX, nSpY, SOMEWHAT );
vLoSN  = ConstructCFuzzySet( LoSNSZ, loSnX, loSnY, VERY ),
swLoSN = ConstructCFuzzySet( LoSNSZ, loSnX, loSnY, SOMEWHAT ),
vLoHT  = ConstructCFuzzySet( LoHTSZ, loHtX, loHtY, VERY );
swLoHT = ConstructCFuzzySet( LoHTSZ, loHtX, loHtY, SOMEWHAT );
# define HTINC 0.01
# define SPINC 1.0
# define SNINC 0.25
 for(ht = 0.02;  ht <= loHtX[LoHTSZ-1];  ht += HTINC) {
  for(Lsp = nSpX[0]; Lsp <= med_sp; Lsp += SPINC) {
   double modLsp = fmod(Lsp,med_sp);
   for(Rsp = nSpX[0]; Rsp <= med_sp; Rsp += SPINC) {
    double modRsp = fmod(Rsp,med_sp);
    for(sn = loSnX[0]; sn <= loSnX[LoSNSZ-1];  sn += SNINC) {
     CFuzzySet *RULE1, *RULE1b, *RULE2, *RULE3, *RULE3b;
     double vloHT,swloHT,nswloHT,
          vnLSP,vnRSP,vnSP,nvnSP,
          swnLSP,swnRSP,swnSP,nswnSP,
          swloSN,nswloSN,vloSN,
          out[ 3 ];
     double maxci = 0.0, ci[3];
     int jdx, maxjdx = 0;
     RULE1 = ConstructCFuzzySet( 2, okxy, okxy,NOHEDGE );
     RULE1b = ConstructCFuzzySet(2, okxy, okxy,NOHEDGE ),
     RULE2 = ConstructCFuzzySet( 2, okxy, okxy,NOHEDGE );
```

```
RULE3 = ConstructCFuzzySet( 2, okxy, okxy,NOHEDGE ),
RULE3b = ConstructCFuzzySet(2, okxy, okxy,NOHEDGE ),
vnLSP  = vNSP->membership( vNSP, modLsp );
vnRSP  = vNSP->membership( vNSP, modRsp );
vnSP = AND(vnLSP,vnRSP);
swnLSP  = swNSP->membership( swNSP, modLsp );
swnRSP  = swNSP->membership( swNSP, modRsp );
  swnSP = AND(swnLSP,swnRSP);
 vloSN  = vLoSN->membership( vLoSN, sn );
swloSN = swLoSN->membership( swLoSN, sn );
 vloHT  = vLoHT->membership( vLoHT, ht );
swloHT = swLoHT->membership( swLoHT, ht );
 nvnSP = (1.0 - vnSP);
nswnSP = (1.0 - swnSP);
nswloSN = (1.0 - swloSN);
nswloHT = (1.0 - swloHT);
 RULE1->scale( RULE1, AND( nswloSN, vnSP ) );
RULE1b->scale( RULE1b, AND( nswloSN, nswloHT ) );
RULE1->cj(RULE1,RULE1b,DISJ);
RULE2->scale( RULE2, AND(vloHT,nswnSP ) );
RULE3->scale( RULE3, AND(vloSN,vnSP) );
RULE3b->scale( RULE3b, AND(vloSN,nswloHT) );
RULE3->cj(RULE3,RULE3b,DISJ);
if(1==RULE1->fcentroid(RULE1,&out[0]))
out[0] = RULE1->membership(RULE1,out[0]);
if(1==RULE3->fcentroid(RULE3,&out[1]))
out[1] = RULE3->membership(RULE3,out[1]);
if(1==RULE2->fcentroid(RULE2,&out[2]))
out[2] = RULE2->membership(RULE2,out[2]);
 printf("\n{ht,Lsp,Rsp,sn}:{%3.2lf,%2.0lf,%2.0lf,%3.1lf} -> ",
 ht,Lsp,Rsp,sn);
```

```
        printf("{OK,N,OMIT}:{%3.2lf,%3.2lf,%3.2lf}", out[0],out[1],out[2]);
        ci[0] = RULE1->compatIndex(RULE1);
        ci[1] = RULE2->compatIndex(RULE2);
        ci[2] = RULE3->compatIndex(RULE3);
        for(jdx = 0; jdx < 3; jdx++)
         if(maxci < ci[jdx]) maxci = ci[maxjdx=jdx];
        printf(" (CI=%3.2lf)",maxci);
#if 0
        if(maxci < 0.5) {
         printf("\n\tLow Compatability Index: CLASS #%d w MAX=%3.2lf",
          ++maxjdx,maxci);
         printf("\n\tvnLSP=%3.2lf vnRSP=%3.2lf vnSP=%3.2lf nvnSP=%3.2lf",
          vnLSP,vnRSP,vnSP,nvnSP);
         printf("\n\tswnLSP=%3.2lf swnRSP=%3.2lf swnSP=%3.2lf nswnSP=%3.2lf",
          swnLSP,swnRSP,swnSP,nswnSP);
         printf("\n\tvloHT=%3.2lf swloHT=%3.2lf nswloHT=%3.2lf",
          vloHT,swloHT,nswloHT);
         printf("\n\tvloSN=%3.2lf swloSN=%3.2lf nswloSN=%3.2lf",
          vloSN,swloSN,nswloSN);
         printf("\n\tRULE1 : !swloSN &          vnSP");
         printf("\n\tRULE1b: !swloSN & !swloHT");
         printf("\n\tRULE2 :          vloHT & !swnSP");
         printf("\n\tRULE3 :  vloSN &          vnSP");
         printf("\n\tRULE3b:  vloSN & !swloHT");
         RULE1->print(RULE1,"RULE1");
         RULE1b->print(RULE1b,"RULE1b");
         RULE2->print(RULE2,"RULE2");
         RULE3->print(RULE3,"RULE3");
         RULE3b->print(RULE3b,"RULE3b");
         vNSP->print(vNSP,"vNSP");
         swNSP->print(swNSP,"swNSP");
```

```
        vLoSN->print(vLoSN,"vLoSN");
        swLoSN->print(swLoSN,"swLoSN");
        vLoHT->print(vLoHT,"vLoHT");
        swLoHT->print(swLoHT,"swLoHT");
        }
#endif
      DestructCFuzzySet(RULE1);
      DestructCFuzzySet(RULE1b);
      DestructCFuzzySet(RULE2);
      DestructCFuzzySet(RULE3);
      DestructCFuzzySet(RULE3b);
      }
     }
    }
  printf("\n");
  }
 DestructCFuzzySet(vNSP);
 DestructCFuzzySet(swNSP);
 DestructCFuzzySet(vLoSN);
 DestructCFuzzySet(swLoSN);
 DestructCFuzzySet(vLoHT);
 DestructCFuzzySet(swLoHT);
}
void
main(int argc, char* argv[]) {
 double normSP;
 if(2 != argc) {
  fprintf(stderr,"usage: %s normSpacing\n",argv[0]);
  exit(1);
  }
 sscanf(argv[1],"%lf",&normSP);
```

```
    ftn_omitokn(normSP);
    exit(0);
  }
  #endif

/*************************************************************************
 * FILE:        CorrCoef.cxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <nrc/nr.hxx>
double
corrcoef( float const* xp, float const* yp, int N )
{
  struct S { double xBar, yBar, xy2, xx2, yy2; } s;
  int idx;
  if(N<2) return 0.0;
  s.xBar = s.yBar = 0.0;
  for(idx=0 ; idx < N; idx++) {
    s.xBar += double(xp[idx]);
    s.yBar += double(yp[idx]);
    }
  s.xBar /= double(N);
  s.yBar /= double(N);
  s.xy2 = s.xx2 = s.yy2 = 0.0;
  for(idx=0 ; idx < N; idx++) {
    double dx, dy;
    dx = (double(xp[idx]) - s.xBar);
    dy = (double(yp[idx]) - s.yBar);
    s.xy2 += dx*dy;
    s.xx2 += dx*dx;
```

```
    s.yy2 += dy*dy;
    }
  s.xy2 /= (double(N)-1.0);
  s.xx2 /= (double(N)-1.0);
  s.yy2 /= (double(N)-1.0);
  if(0.0==s.xx2 || 0.0==s.yy2)
    return 0.0;
  else
    return s.xy2/::sqrt(s.xx2*s.yy2);
}

/*****************************************************************************
 * FILE: fuzzyset.h
 * Copyright © 1996, University of Utah
 */
#if !defined(_FUZZYSET_H_)
#define _FUZZYSET_H_
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#if defined(__cplusplus)
extern "C" {
#endif
typedef enum { DISJ, CONJ } LOGICAL_OPERATOR;
typedef enum { NOHEDGE, SOMEWHAT, VERY, CONTINT } HEDGE;
#define AND(v1,v2) ((v1<v2)?(v1):(v2))
#define OR(v1,v2) ((v1>v2)?(v1):(v2))
#define NOT(v)   (1-(v))
typedef struct CFuzzySet {
  int (*fcentroid)( struct CFuzzySet const* pcfz, double* pcentroid );
```

```
    void (*cj)( struct CFuzzySet* pcfz,struct CFuzzySet const* ps,LOGICAL_OPERATOR cjdj
  );
    int (*invalid)( struct CFuzzySet const* pcfz );
    double (*membership)( struct CFuzzySet const* pcfz, double pt );
    double (*compatIndex)( struct CFuzzySet const* pcfz );
    void (*negate)( struct CFuzzySet* pcfz );
    void (*print)( struct CFuzzySet const* pcfz, char const* pname );
    void (*scale)( struct CFuzzySet* pcfz,  double fac );
    int n;
    double* x;
    double* y;
    HEDGE d_ht;
    double (*d_phedge)(double yval);
    double (*d_punhedge)(double yval);
    } CFuzzySet;
  CFuzzySet*
   ConstructCFuzzySet(int npts,double const* xpts,double const* ypts,HEDGE ht);
  void DestructCFuzzySet(CFuzzySet* pcfz);
  #if defined(__cplusplus)
  }
  #endif
  #endif

  /*****************************************************************************
   * FILE:       gaussj.cxx
   * TYPIST:     Andy Marks
   * Copyright © 1996 University of Utah
   */
  #include <math.h>
  #include <nrc/nrutil.hxx>
  #define SWAP(a,b) { temp=(a); (a)=(b); (b)=temp; }
```

```
#if !defined(SA)
void
gaussj(float** a, int n, float** b, int m)
{
  int *indxc, *indxr, *ipiv;
  int i, icol, irow,j,k,el,ll;
  float pivinv, temp;
  indxc = ivector(1,n);
  indxr = ivector(1,n);
  ipiv  = ivector(1,n);
  for(j=1;j<=n;j++) ipiv[j] = 0;
  for(i=1;i<=n;i++) {
   float big = 0.0f;
   for(j=1;j<=n;j++)
    if(1 != ipiv[j])
     for(k=1;k<=n;k++) {
      if(0==ipiv[k]) {
       if(fabs(a[j][k]) >= big) {
        big = float(fabs(a[j][k]));
        irow = j;
        icol = k;
       }
      }
     else if(ipiv[k] > 1)
      nrerror( "gauusj: Singular Matrix-1" );
     }
   ++(ipiv[icol]);
   if(irow != icol) {
    for(el=1;el<=n;el++) SWAP(a[irow][el],a[icol][el]);
    for(el=1;el<=m;el++) SWAP(b[irow][el],b[icol][el]);
    }
```

```
        indxr[i]=irow;
        indxc[i]=icol;
        if(0.0f == a[icol][icol])
          nrerror("gaussj: Singular Matrix-2");
        pivinv = 1.0f/a[icol][icol];
        a[icol][icol]=1.0f;
        for(el=1;el<=n;el++) a[icol][el] *= pivinv;
        for(el=1;el<=m;el++) b[icol][el] *= pivinv;
        for(ll = 1; ll <=n; ll++)
          if(ll != icol) {
            float dum = a[ll][icol];
            a[ll][icol]=0.0f;
            for(el=1;el<=n;el++) a[ll][el] -= a[icol][el]*dum;
            for(el=1;el<=m;el++) b[ll][el] -= b[icol][el]*dum;
            }
        }
       for(el = n; el>=1; el--) {
        if(indxr[el] != indxc[el])
          for(k=1;k<=n;k++)
            SWAP( a[k][indxr[el]], a[k][indxc[el]] );
        }
       free_ivector(ipiv,1,n);
       free_ivector(indxr,1,n);
       free_ivector(indxc,1,n);
}
#endif
#if defined(SA)
#include <stdio.h>
#include <nrc/nr.hxx>
int
main(int argc, char* argv[])
```

```
{
 float **a, **b, **aorig, **borig, **aprod, **bprod;
 int r, c, s;
 a = matrix(1,4,1,4);
 aorig = matrix(1,4,1,4);
 aprod = matrix(1,4,1,4);
 b = matrix(1,4,1,2);
 borig = matrix(1,4,1,2);
 bprod = matrix(1,4,1,2);
 a[1][1] =  7.0f; a[1][2] =  8.0f; a[1][3] =  9.0f; a[1][4] = 10.0f;
 a[2][1] =  6.0f; a[2][2] =  1.0f; a[2][3] =  2.0f; a[2][4] = 11.0f;
 a[3][1] =  5.0f; a[3][2] =  4.0f; a[3][3] =  3.0f; a[3][4] = 12.0f;
 a[4][1] = 16.0f; a[4][2] = 15.0f; a[4][3] = 14.0f; a[4][4] = 13.0f;
 for(r=1; r<=4; r++)
   for(c=1; c<=4; c++)
     aorig[r][c] = a[r][c];
 b[1][1] =  50.0f; b[1][2] =  40.0f;
 b[2][1] = 122.0f; b[2][2] =  96.0f;
 b[3][1] = 194.0f; b[3][2] = 152.0f;
 b[4][1] = 266.0f; b[4][2] = 208.0f;
 for(r=1; r<=4; r++)
   for(c=1; c<=2; c++)
     borig[r][c] = b[r][c];
 gaussj( a, 4, b, 2);
 printf("inv(A):\n");
 for(r=1;r<=4;r++) {
   for(c=1;c<=4;c++)
     printf("%9.4f ",a[r][c]);
   printf("\n");
   }
 printf("S:\n");
```

```
for(r=1;r<=4;r++) {
 printf("\t");
 for(c=1;c<=2;c++)
  printf("%9.4f ",b[r][c]);
 printf("\n");
 }
for(r=1;r<=4;r++)
 for(c=1;c<=4;c++) {
  float sum = 0.0f;
  for(s=1;s<=4;s++)
   sum += a[r][s]*aorig[s][c];
  aprod[r][c] = sum;
  }
printf("inv(A)*A==I?\n");
for(r=1;r<=4;r++) {
 printf("\t");
 for(c=1;c<=4;c++)
  printf("%9.4f ",aprod[r][c]);
 printf("\n");
 }
for(r=1;r<=4;r++)
 for(c=1;c<=2;c++) {
  float sum = 0.0f;
  for(s=1;s<=4;s++)
   sum += aorig[r][s]*b[s][c];
  bprod[r][c] = sum;
  }
printf("A*S==B?\n");
for(r=1;r<=4;r++) {
 printf("\t");
 for(c=1;c<=2;c++)
```

```
      printf("%9.4f ",bprod[r][c]);
     printf("\n");
     }
    for(r=1;r<=4;r++)
     for(c=1;c<=2;c++) {
      float sum = 0.0f;
      for(s=1;s<=4;s++)
       sum += a[r][s]*borig[s][c];
      bprod[r][c] = sum;
      }
    printf("inv(A)*B==S?\n");
    for(r=1;r<=4;r++) {
     printf("\t");
     for(c=1;c<=2;c++)
      printf("%9.4f ",bprod[r][c]);
     printf("\n");
     }
    free_matrix(a,1,4,1,4);
    free_matrix(aorig,1,4,1,4);
    free_matrix(aprod.1.4,1,4);
    free_matrix(b,1,4,1,2);
    free_matrix(borig,1,4,1,2);
    free_matrix(bprod.1,4,1,2);
    return 0;
}
#endif

/*************************************************************************
 * FILE:        mb.cxx -- stands for "MyBusiness"
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
```

```
 */
#include <basecall/mb.hxx>
#undef AS_WAS
static int const BGNLFTR=13,
                 ENDLFTR=23;
MB::MB( int fluor ) : minFreq_(1.0), minFreqLane_(0), lastFbw_(0)
{
  bdStatics( fluor );
  if(fluor) {
    static double const CROSSOVER = 0.23;
    static int const MAXFBW = 100;
    double K = 2.0*::sqrt(::log(CROSSOVER) / -0.5);
    for(int pdx=SMLGAP;pdx<=BIGGAP;pdx++) {
      fbwlut_[pdx] = int( 0.5 + (K/double(pdx))*double(NPTS)/(2.0*NR_PI) );
      if(fbwlut_[pdx] > MAXFBW)
        fbwlut_[pdx] = MAXFBW;
      }
    }
  else {
    static double const CROSSOVER = 0.20;
    static int const MAXFBW = 100;
    double K = 2.0*::sqrt(::log(CROSSOVER) / -0.5);
    for(int pdx=SMLGAP;pdx<=BIGGAP;pdx++) {
      fbwlut_[pdx] = int( 0.5 + (K/double(pdx))*double(NPTS)/(2.0*NR_PI) );
      if(fbwlut_[pdx] > MAXFBW)
        fbwlut_[pdx] = MAXFBW;
      }
    }
}
MB::~MB()
{
```

```
}
MB::MB( MB const& rhs ) : minFreq_(1.0), minFreqLane_(0), lastFbw_(0)
{
  *this = rhs;
}
MB const&
MB::operator=(MB const& rhs)
{
  int idx, jdx;
  lastFbw_    = rhs.lastFbw_;
  for(idx=jdx=1;idx<=NPTS;idx+=1,jdx+=2) {
   ivec_[jdx]   = rhs.ivec_[jdx]; ivec_[jdx+1] = rhs.ivec_[jdx+1];
   imag_[jdx]   = rhs.imag_[jdx]; imag_[jdx+1] = rhs.imag_[jdx+1];
   lifter_[idx] = rhs.lifter_[idx];
   filter_[idx] = rhs.filter_[idx];
   ww_[idx]     = rhs.ww_[idx];
   }
  minFreq_    = rhs.minFreq_;
  minFreqLane_ = rhs.minFreqLane_;
  for(idx=0;idx<5;idx++)
   bandAmpl_[idx] = rhs.bandAmpl_[idx];
  for(idx=SMLGAP;idx<=BIGGAP;idx++)
   fbwlut_[idx] = rhs.fbwlut_[idx];
  return *this;
}
void
MB::bdStatics( int fluor )
{
  double bgnpt, endpt, m, b;
  int sdx;
  lastFbw_ = 0;
```

```
        lifter_[1] = lifter_[NPTS] = 0.0;
        lifter_[NPTS/2+1] = 1.0;
        if(fluor) { bgnpt = 7;  endpt = 24; }
        else      { bgnpt = BGNLFTR; endpt = ENDLFTR; }
        m = NR_PI/(endpt-bgnpt);
        b = NR_PI/2.0 - m*endpt;
        for(sdx=2;sdx<=NPTS/2;sdx++) {
         if(sdx < bgnpt)
          lifter_[sdx] = 0.0;
         else if(sdx >= bgnpt && sdx <= endpt)
          lifter_[sdx] = 0.5 * (1.0 + ::sin( m*double(sdx) + b ));
         else
          lifter_[sdx] = 1.0;
         lifter_[NPTS-sdx+2] = lifter_[sdx];
         }
        for(int j=1; j<=NPTS; j++) {
         ww_[j] = 2.0*NR_PI/double(NPTS) * (double(j) - double(NPTS/2));
         ww_[j] *= ww_[j];
         }
    }
    int
    MB::remune() const
    {
    #if 1
     return 1;
    #else
      static int const OK_YEAR  = 96;
      time_t now = time((time_t*)NULL);
      struct tm* ptm = localtime( &now );
      if(OK_YEAR != ptm->tm_year)
       return 0;
```

```
    else if(ptm->tm_mon>=3 && ptm->tm_mon<=11)
     return 1;
    else
     return 0;
#endif
}

/*************************************************************************
 * FILE:        mb.hxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#ifndef _MB_HXX_
#define _MB_HXX_
#include <basecall/Pkdet.hxx>
#include <basecall/seqrdr.hxx>
#include <basecall/SegRead.hxx>
#if defined(sun)||defined(_WIN32)
# define nint(v) int(((v)>0)?((v)+0.5):((v)-0.5))
#endif
#if defined(_WIN32)
 double drand48();
# define isnan  _isnan
# define finite _finite
#endif
static const int SMLGAP = 1,
            BIGGAP = 33;
class MB
{
public:
   enum Status { STS_UNINITD, STS_INITD, STS_NO_MEM };
```

```
    MB( int fluor );
    MB( MB const& rhs );
    MB const& operator=(MB const& rhs);
    ~MB();
    int remune() const;
    double  minFreq_,
         bandAmpl_[5];
    int    minFreqLane_;
    double  ivec_[1+2*NPTS],
         imag_[1+2*NPTS],
         filter_[1+NPTS],
         lifter_[1+NPTS],
         ww_[1+NPTS];
    int    lastFbw_;
    int    fbwlut_[1+BIGGAP];
private:
    void bdStatics( int fluor );
    Status status_;
};
#endif

/*****************************************************************************
 * FILE:        Metrics.hxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#ifndef _METRICS_HXX_
#define _METRICS_HXX_
#if defined(WIN32)
class _declspec( dllexport) BandStat
#else
```

```
class BandStat
#endif
{
public:
    BandStat();
    ~BandStat() {};
    void ntnr( int v )   { ntnr_ = v; }
    void posn( int v )   { posn_ = v; }
    void hght( float v ) { hght_ = v; }
    void lowv( float v ) { lowv_ = v; }
    void xbnd( float v ) { xbnd_ = v; }
    void shap( float v ) { shap_ = v; }
    void widt( float v ) { widt_ = v; }
    void lgap( float v ) { lgap_ = v; }
    void sgap( float v ) { sgap_ = v; }
    void buzz( float v ) { buzz_ = v; }
    void bbgn( int v)    { bbgn_ = v; }
    void bend( int v)    { bend_ = v; }
    void insr( int v )   { insr_ = v; }
    void call( char v )  { call_ = v; }
    void qual( float v ) { qual_ = v; }
    int ntnr()   const { return ntnr_; }
    int posn()   const { return posn_; }
    float hght() const { return hght_; }
    float lowv() const { return lowv_; }
    float xbnd() const { return xbnd_; }
    float shap() const { return shap_; }
    float widt() const { return widt_; }
    float lgap() const { return lgap_; }
    float sgap() const { return sgap_; }
    float buzz() const { return buzz_; }
```

```
    int  bbgn() const { return bbgn_; }
    int  bend() const { return bend_; }
    int  insr() const { return insr_; }
    int  awid() const { return bend_-bbgn_+1; }
    float qual() const { return qual_; }
    float StadenQual() const { return 98.0f*qual_+1.0f; }
    char  call() const { return call_; }
    void debug() const;
  private:
    int  ntnr_;
    int  posn_;
    int  bbgn_;
    int  bend_;
    int  insr_;
    float hght_;
    float lowv_;
    float xbnd_;
    float shap_;
    float buzz_;
    float widt_;
    float lgap_;
    float sgap_;
    char  call_;
    float qual_;
};
#if defined(WIN32)
class _declspec( dllexport) BandStatArray
#else
class BandStatArray
#endif
{
```

```
public:
  BandStatArray();
  BandStatArray( BandStatArray const& rhs );
  BandStatArray const& operator=( BandStatArray const& rhs );
  ~BandStatArray();
  void init( int len );
  void ntnr( int idx, int v )   { band_[ idx ].ntnr(v); }
  void posn( int idx, int v )   { band_[ idx ].posn(v); }
  void bbgn( int idx, int v )   { band_[ idx ].bbgn(v); }
  void bend( int idx, int v )   { band_[ idx ].bend(v); }
  void insr( int idx, int v )   { band_[ idx ].insr(v); }
  void hght( int idx, float v ) { band_[ idx ].hght(v); }
  void lowv( int idx, float v ) { band_[ idx ].lowv(v); }
  void xbnd( int idx, float v ) { band_[ idx ].xbnd(v); }
  void shap( int idx, float v ) { band_[ idx ].shap(v); }
  void buzz( int idx, float v ) { band_[ idx ].buzz(v); }
  void widt( int idx, float v ) { band_[ idx ].widt(v); }
  void lgap( int idx, float v ) { band_[ idx ].lgap(v); }
  void sgap( int idx, float v ) { band_[ idx ].sgap(v); }
  void qual( int idx, float v ) { band_[ idx ].qual(v); }
  void call( int idx, char v )  { band_[ idx ].call(v); }
  void append( BandStatArray const& rhs, int oselend, int nselbgn );
  int   ntnr( int idx ) const { return band_[idx].ntnr(); }
  int   posn( int idx ) const { return band_[idx].posn(); }
  int   bbgn( int idx ) const { return band_[idx].bbgn(); }
  int   bend( int idx ) const { return band_[idx].bend(); }
  int   insr( int idx ) const { return band_[idx].insr(); }
  float hght( int idx ) const { return band_[idx].hght(); }
  float lowv( int idx ) const { return band_[idx].lowv(); }
  float xbnd( int idx ) const { return band_[idx].xbnd(); }
  float shap( int idx ) const { return band_[idx].shap(); }
```

```
    float buzz( int idx ) const { return band_[idx].buzz(); }
    float widt( int idx ) const { return band_[idx].widt(); }
    float lgap( int idx ) const { return band_[idx].lgap(); }
    float sgap( int idx ) const { return band_[idx].sgap(); }
    float qual( int idx ) const { return band_[idx].qual(); }
    float StadenQual(int idx) const { return band_[idx].StadenQual(); }
    char  call( int idx ) const { return band_[idx].call(); }
    int   len()         const { return len_; }
    BandStat&      band( int idx )      { return band_[idx]; }
    BandStat const& band( int idx ) const { return band_[idx]; }
    void debug() const;
private:
    int len_;
    BandStat* band_;
};
#endif

/*****************************************************************************
* FILE:         nfeeder.cxx
* AUTHOR:   Andy Marks
* COPYRIGHT (c) 1996, University of Utah
*/
#include <basecall/mb.hxx>
#include <nrc/Centroid.hxx>
static const int
  MAXPASSES = 6,
  INPUTSTEP = 1900,
  ENDPT    = (INPUTSTEP+NPTS)/2,
  FROMEND  = (NPTS-INPUTSTEP)/10;
static const int   PERCENTILE = 40;
```

```
static int
i_cmp( void const* e1, void const* e2 )
{
  return (*(int*)e1) - (*(int*)e2);
}
int
SegRead::fBandSpace( int& spacing ) const
{
  BandStatArray const& m = bandStats();
  int len, idx0,idx1;
  int* diff;
  if(m.len()<2)
   return 0;
  len = int( m.len()-1 );
  if(NULL == (diff = ::ivector( 1, len )))
   return 0;
  for(idx0=0,idx1=1; idx0<len; idx0++,idx1++)
   diff[idx1] = m.posn(idx1) - m.posn(idx0);
  ::qsort( &diff[1], len, sizeof(*diff), i_cmp );
  spacing = diff[(PERCENTILE*len)/100];
  ::free_ivector( diff, 1, len );
  return 1;
}
int
Wvfm::nfeeder( RdrOut& output, int fVb )
{
  int PASSES, rawpts, fbw=82, spacing;
  int rv = 0;
  srand(0);
  output.qualctrl().startTimer();
  if(1 != preproc()) {
```

```
 if(fVb) ::fprintf(stderr,"preproc() failed\n");
 return 0;
 }
output.iS1( bgni() );
rawpts = endi()-bgni()+1;
if(rawpts < NPTS)
 PASSES = 1;
else {
 PASSES = int(ceil(float(rawpts-NPTS)/float(INPUTSTEP)) + 1);
 if(PASSES > MAXPASSES)
  PASSES = MAXPASSES;
 }
SegRead segrd( bgni(), FLUOR==ds() );
for(int passNr=1; passNr <= PASSES; passNr++) {
 if(0 == (rv = nreader( fbw, 0, segrd ))) {
  if(fVb) ::fprintf(stderr,"nreader(0) failed,passNr=%d\n",passNr);
  break;
  }
 else if(0 == (rv = segrd.fBandSpace( spacing ))) {
  if(fVb) ::fprintf(stderr,"fBandSpace(%d) failed,passNr=%d\n",passNr);
  break;
  }
 else {
  if(1!=passNr) {
   int ospace = output.qualctrl().bspac(passNr-2);
   if(spacing < ospace)
    spacing = ospace;
   }
  segrd.fbwlut( spacing, fbw );
  if(0 == (rv = nreader( fbw, 1, segrd ))) {
   if(fVb) ::fprintf(stderr."nreader(1) failed,passNr=%d\n",passNr);
```

```
        break;
        }
       else if(0 == (rv = output.add( passNr, fbw, spacing, segrd ))) {
        if(fVb) ::fprintf(stderr,"output.add() failed,passNr=%d\n",passNr);
        rv = 1;
        break;
        }
       }
      int newStart = segrd.iS1()+INPUTSTEP;
      if((newStart+NPTS) >= endi())
       newStart = endi()-NPTS;
      segrd.iS1( newStart );
      }
    output.qualctrl().stopTimer();
    return rv;
}

/*****************************************************************************
 * FILE:        nr.hxx
 * TYPIST:    Andy Marks
 * Copyright © 1996 University of Utah
 */
#ifndef _NR_HXX_
#define _NR_HXX_
#include <math.h>
#include <nrc/nrutil.hxx>
#if defined(__cplusplus)
 static double const NR_PI = acos(-1.0);
#else
# define NR_PI acos(-1.0)
#endif
```

```
enum PFWGHT { INSTRUMENTAL=-1, NO_WEIGHTING, STATISTICAL };
DLLexport int polfit(float const* x, float const* y, float const* sigmaY,
                     int NPTS, int NTERMS, PFWGHT wm,
                float coef[], float& chisq);
DLLexport int ilinreg(int const* x, int const* y, int n, float coef[3], float* rr );
DLLexport int linreg(float const* x, float const* y, int n, float coef[3], float* rr );
DLLexport double corrcoef(float const* v1, float const* v2, int N);
DLLexport int iquadratic(int const* x, int const* y, int n, float coef[4]);
DLLexport int dquadratic(double const* x, double const* y, int n, float coef[4]);
DLLexport void gaussj(float** a, int n, float** b, int m);
DLLexport void spline(float const x[], float const y[], int n, float yp1, float ypn, float y2[]);
DLLexport void splint(float const xa[],float const ya[],float const ya2[],int n,
                float x, float *y);
DLLexport void dfour1(double data[], unsigned long nn, int isign);
#endif
/*************************************************************************
 * FILE:        nreader.cxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
#if defined(_WIN32)
double
drand48()
{
        return double(rand()) / double(RAND_MAX);
}
#endif
void
randPadd( double** rm, int rows, int cols )
{
```

```
    for(int r=1; r<=rows; r++)
     for(int c=1; c<=cols; c++)
      rm[r][c] = drand48();
}
double
dmean( double* v, int b, int e )
{
   double sum = 0.0;
   for(int idx=b; idx<=e; idx++)
     sum += v[idx];
   return sum/double(e-b+1);
}
int
d_cmp( void const* e1, void const* e2 )
{
   double v1 = (*(double const*)e1),
        v2 = (*(double const*)e2);
   if(v1>v2)
    return 1;
   else if(v1<v2)
    return -1;
   else
    return 0;
}
int
Wvfm::nreader( int FBW, int pass2, SegRead& sr )
{
   Wvfm bdproc( NPTS, 4, lnordr(), ds(), method() );
   int have, endpt, iSN, scnl, lane, pt1 = sr.iS1();
   iSN = endi();
   endpt = pt1+NPTS-1;
```

```
        if(endpt <= iSN) {
         have = NPTS;
         for(lane=1;lane<=4;lane++)
          for(scnl=pt1;scnl<=endpt;scnl++)
           bdproc.sc_la_set(scnl-pt1+1,lane,sc_la(scnl,lane));
        }
        else {
         double **padding, *v;
         int need = endpt - iSN, q5, q95;
#        undef  PADMATRIX
#        define  PADVECT
#        if defined(PADVECT)
          int k=1, *pl = &k,   NPLN=1;
#        else
          int     *pl = &lane, NPLN=4;
#        endif
         if(NULL == (padding = ::dmatrix(1,10+need,1,NPLN))) {
          status_ = STS_NO_MEM;
          return 0;
          }
         ::randPadd( padding, 10+need, NPLN );
         have = iSN-pt1+1;
         q5 = int(nint(float(have)/20.0f));
         q95 = have-q5;
         v = ::dvector( 1, have );
         for(lane=1; lane<=4; lane++) {
          double ht;
          for(scnl=1; scnl<=have; scnl++) {
           v[scnl] = sc_la( pt1+scnl-1 , lane );
           bdproc.sc_la_set(scnl,lane,v[scnl]);
           }
```

```
::qsort( &v[1], have, sizeof(double), d_cmp );
ht = v[q95]-v[q5];
for(scnl=1; scnl<=10; scnl++) {
 double v = bdproc.sc_la(have-(10-scnl),lane),
      p = ht*padding[scnl][*pl],
      vm = 0.5*(1.0+cos(double(scnl)*NR_PI/10.0)),
      vp = 0.5*(1.0+cos(NR_PI+double(scnl)*NR_PI/10.0));
  bdproc.sc_la_set(scnl,lane,vm*v + vp*p);
 }
 for(scnl=1; scnl<=need; scnl++)
  bdproc.sc_la_set(have+scnl,lane,ht*padding[10+scnl][*pl]);
 }
::free_dmatrix( padding, 1,10+need, 1,NPLN );
::free_dvector( v, 1,have );
}
Wvfm source( bdproc );
sr.blindeconv( bdproc, FBW );
if(1 != sr.xtranorm( bdproc, source, have, pass2 ))
 return 0;
Wvfm aligned( NPTS+sr.nsv().maxshft(), 4, lnordr(), ds(), method() );
aligned.ssm( ssm() );
for(lane=1; lane<=4; lane++) {
 short sv = sr.nsv().s(lane);
 for(scnl=1; scnl<=NPTS; scnl++)
  aligned.sc_la_set( sv+scnl, lane, bdproc.sc_la(scnl,lane) );
 }
if(pass2)
 aligned.endi( have+sr.nsv().maxshft() );
sr.wvfm( aligned );
ShftVect noshft;
sr.wvfm()->envelope( noshft );
```

```
  return sr.nrefine( lnordr(), FBW, have, pass2 );
}

/*****************************************************************************
 * FILE:        nrefine.cxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
#include <basecall/fgapcheck.h>
int
insMetric( int const* px, int const* py, int* ytmp, int N )
{
  float coef[4], std;
  int idx, jdx, *xtmp = NULL;
  if(N<4)
   return 0;
  if(1 != ::iquadratic( px,py,N, coef ))
   return 0;
  std = float( ::sqrt( coef[3] ) );
  xtmp = ::ivector(1,N);
  if(NULL == xtmp) return 0;
  for(jdx=idx=1; idx<=N; idx++) {
   float x = float(px[idx]);
   ytmp[jdx] = int(coef[0]+x*coef[1]+x*x*coef[2]);
   xtmp[jdx] = px[idx];
   if(abs(ytmp[jdx]-py[idx]) < std)
    jdx++;
   }
  if(--jdx >= 4) {
   if(1 != ::iquadratic( xtmp,ytmp,jdx, coef )) {
```

```
        ::free_ivector(xtmp,1,N);
        return 0;
        }
       for(idx=1; idx<=N; idx++) {
        float x = float(px[idx]);
        ytmp[idx] = int(coef[0]+x*coef[1]+x*x*coef[2]);
        }
       }
      ::free_ivector(xtmp,1,N);
      if(0.0f != coef[2]) {
       int Ipt = (int)nint(-coef[1]/(2.0*coef[2]));
       if(Ipt>=1 && Ipt<=N) {
        int K = ytmp[Ipt];
        if(coef[2] > 0.0)
         for(idx=1;idx<Ipt;idx++)
          ytmp[idx] = K;
        else for(idx=(Ipt+1);idx<=N;idx++)
         ytmp[idx] = K;
        }
       }
      return 1;
}
int
SegRead::centroid_( int bgn, int end ) const
{
   Wvfm const& w = *wvfm();
   double numer=0.0, denom=0.0;
   int idx=bgn+1, jdx=idx-1, N = end-bgn+1;
   if(N < 2)
    return (bgn+N/2);
   else {
```

```
        double minv = 100000.0;
        for(int kdx=bgn;kdx<=end;kdx++)
         if(w.envv(kdx)<minv)
          minv = w.envv(kdx);
        for( ;idx<=end; idx++,jdx++) {
         double vj = w.envv(jdx)-minv;
         double vi = w.envv(idx)-minv;
         numer += vj*(2.0*jdx + idx) + vi*(jdx + 2.0*idx);
         denom += (vj + vi);
         }
        }
       return int(nint(numer/(3.0*denom)));
}
int
SegRead::nrefine( char const* LNORDR, int FBW, int npts, int pass2 )
{
   PKDET rawPks,
       refPks;
   float *xbnd, *ht, *lo, **om;
   int *bandcode, *insSP, *insWD, NPK, idx, jdx, sts;
   if(1 != peakdet( npts, rawPks ))
    return 0;
   NPK = rawPks.npk();
   xbnd = xbndara_( rawPks );
   if(NULL == xbnd) {
    status_ = STS_NO_MEM;
    return 0;
    }
   if(NULL == (bandcode = ::ivector(1, nWvf_->scanl() ))) {
    status_ = STS_NO_MEM;
    ::free_vector( xbnd,1.NPK );
```

```
    return 0;
    }
  maxlanecode_( npts, bandcode );
  if(NPK >= STATIC_BUF_SZ) {
    ::free_vector( xbnd,1,NPK );
    status_ = STS_BUF2SMALL;
    return 0;
    }
  insSP = ::ivector( 1, NPK );
  if(NULL == insSP) {
    status_ = STS_NO_MEM;
    ::free_vector( xbnd,1,NPK );
    return 0;
    }
  if(1 != ::insMetric( rawPks.bmid(), rawPks.lgap(), insSP, NPK )) {
    status_ = STS_TOO_FEW;
    ::free_vector( xbnd,1,NPK );
    ::free_ivector(insSP,1,NPK);
    return 0;
    }
  ht = ::vector( 1, NPK );
  if(NULL == ht) {
    status_ = STS_NO_MEM;
    ::free_vector( xbnd,1,NPK );
    ::free_ivector( insSP, 1, NPK );
    return 0;
    }
  for(idx = 1; idx <= NPK; idx++)
    ht[ idx ] = float(nWvf_->envv( rawPks.bmid(idx) ));
  lo = ::vector( 1, NPK );
  if(NULL == lo) {
```

```
status_ = STS_NO_MEM;
::free_vector( xbnd,1,NPK );
::free_ivector( insSP, 1, NPK );
::free_vector( ht, 1, NPK );
return 0;
}
for(idx = 1; idx <= NPK; idx++) {
float onsv, ofsv;
onsv = float(nWvf_->envv( rawPks.bbgn(idx) ));
ofsv = float(nWvf_->envv( rawPks.bend(idx) ));
lo[ idx ] = onsv<ofsv?onsv:ofsv;
}
om = ::matrix(1,NPK,1,2);
sts = ::omitokn(insSP,ht,lo,rawPks.lgap(),rawPks.rgap(),xbnd,NPK,om);
::free_vector( xbnd,   1,NPK ); xbnd  = NULL;
::free_vector( ht,     1,NPK ); ht    = NULL;
::free_vector( lo,     1,NPK ); lo    = NULL;
::free_ivector( insSP, 1,NPK ); insSP = NULL;
if(1 != sts) {
::free_matrix( om, 1,NPK,1,2 );
status_ = STS_TOO_FEW;
return 0;
}
enum OKNOMIT { BAND_OK = 1, BAND_N = 2, BAND_OMIT = 3 };
for(jdx=idx=1;idx<=NPK;idx++)
switch(OKNOMIT(nint(om[idx][1]))) {
case BAND_N:
bandcode[ rawPks.bmid( idx ) ] = 5;
case BAND_OK:
nband_[ jdx ] = rawPks.band( idx );
seq_[ jdx ] = LNORDR[ nWvf_->envi( rawPks.bmid( idx ) )-1 ];
```

```
      jdx++;
      break;
    }
  ::free_matrix( om, 1,NPK,1,2 ); om = NULL;
  NPK = jdx-1;
  refPks.set( nband_, NPK );
  insSP = ::ivector( 1, NPK );
  if(NULL == insSP) {
   status_ = STS_NO_MEM;
   return 0;
   }
  if(1 != ::insMetric( refPks.bmid(), refPks.lgap(), insSP, NPK )) {
   status_ = STS_TOO_FEW;
   ::free_ivector( insSP, 1, NPK );
   return 0;
   }
  insWD = ::ivector( 1, NPK );
  if(NULL == insWD) {
   status_ = STS_NO_MEM;
   ::free_ivector( insSP,1,NPK );
   return 0;
   }
  if(1 != ::insMetric( refPks.bmid(), refPks.bwid(), insWD, NPK )) {
   status_ = STS_TOO_FEW;
   ::free_ivector( insSP,1,NPK );
   ::free_ivector( insWD,1,NPK );
   return 0;
   }
  om = ::matrix(1,NPK,1,2);
  sts = ::gapcheck(insSP, insWD, refPks.lgap(),refPks.bwid(),seq_, NPK,om);
  ::free_ivector( insWD, 1,NPK ); insWD = NULL;
```

```
if(1 != sts) {
 ::free_ivector( insSP,1,NPK);
 ::free_matrix(om,1,NPK,1,2);
 return 0;
 }
nband_[ 1 ] = refPks.band( 1 );
for(jdx=idx=2; idx<=NPK; idx++) {
 if(GAP_SPLIT == GPCHK(nint(om[idx][1]))) {
  int m4, numSmallGap, newSP, mid;
  double ratio;
  m4 = idx-1;
  if(0 == insSP[m4]) {
   status_ = STS_TOO_FEW;
   ::free_ivector( insSP,1,NPK );
   ::free_matrix( om,1,NPK,1,2 );
   return 0;
   }
  ratio = double(refPks.rgap(m4))/double(insSP[m4]);
  numSmallGap = int(0.5 + 0.2 + ratio);
  if(0 == numSmallGap) {
   status_ = STS_TOO_FEW;
   ::free_ivector( insSP,1,NPK );
   ::free_matrix( om,1,NPK,1,2 );
   return 0;
   }
  newSP = refPks.rgap( m4 ) / numSmallGap;
   mid = refPks.bmid( m4 );
  for(int ndx=1; ndx<numSmallGap; ndx++) {
   int bgn,end,c;
   mid += newSP;
   bgn = mid-newSP/2;
```

```
        end = bgn+newSP-1;
            Band b( bgn, c=centroid_(bgn,end), end, 1 );
        nband_[ jdx++ ] = b;
        if(jdx >= STATIC_BUF_SZ) {
          status_ = STS_BUF2SMALL;
          ::free_ivector( insSP,1,NPK );
          ::free_matrix( om,1,NPK,1,2 );
          return 0;
          }
        }
      }
     nband_[ jdx++ ] = refPks.band( idx );
     if(jdx >= STATIC_BUF_SZ) {
      status_ = STS_BUF2SMALL;
      ::free_ivector( insSP,1,NPK );
      ::free_matrix( om,1,NPK,1,2 );
      return 0;
      }
     }
    ::free_ivector( insSP, 1,NPK ); insSP = NULL;
    ::free_matrix( om, 1,NPK,1,2 ); om    = NULL;
    NPK = jdx-1;
    refPks.set( nband_, NPK );
    insSP = ::ivector( 1, NPK );
    if(NULL == insSP) {
     status_ = STS_NO_MEM;
     return 0;
     }
    if(1 != ::insMetric( refPks.bmid(), refPks.lgap(), insSP, NPK )) {
     status_ = STS_TOO_FEW;
     ::free_ivector( insSP, 1, NPK );
```

```
    return 0;
    }
  xbnd = xbndara_( refPks );
  if(NULL == xbnd) {
    status_ = STS_NO_MEM;
    ::free_ivector( insSP, 1, NPK);
    return 0;
    }
  ht = ::vector( 1, NPK );
  if(NULL == ht) {
    status_ = STS_NO_MEM;
    ::free_ivector( insSP,1,NPK );
    ::free_vector(  xbnd,1,NPK );
    return 0;
    }
  for(idx=1;idx<=NPK;idx++)
    ht[ idx ] = float(nWvf_->envv( refPks.bmid(idx) ));
  lo = ::vector( 1, NPK );
  if(NULL == lo) {
    status_ = STS_NO_MEM;
    ::free_ivector( insSP,1,NPK );
    ::free_vector(  xbnd,1,NPK );
    ::free_vector(  ht,1,NPK );
    return 0;
    }
  for(idx = 1; idx <= NPK; idx++) {
    float onsv, ofsv;
    onsv = float(nWvf_->envv( refPks.bbgn(idx) ));
    ofsv = float(nWvf_->envv( refPks.bend(idx) ));
    lo[ idx ] = onsv<ofsv?onsv:ofsv;
    }
```

```
om = ::matrix(1,NPK,1,2);
sts = ::omitokn(insSP,ht,lo,refPks.lgap(),refPks.rgap(),xbnd,NPK,om );
::free_vector( xbnd, 1,NPK ); xbnd = NULL;
::free_vector( ht, 1,NPK ); ht = NULL;
::free_vector( lo, 1,NPK ); lo = NULL;
::free_ivector( insSP, 1,NPK ); insSP = NULL;
if(1 != sts) {
 ::free_matrix(om,1,NPK,1,2);
 return 0;
 }
for(idx=jdx=1; idx<=NPK; idx++) {
 int bmid = refPks.bmid(idx);
 switch(OKNOMIT( nint(om[idx][1]) )) {
  case BAND_N:
   bandcode[ bmid ] = 5;
  case BAND_OK:
   nband_[ jdx ] = refPks.band(idx);
    seq_[ jdx ] = LNORDR[ bandcode[ bmid ]-1 ];
   jdx++;
   break;
  default:
   break;
  }
 }
::free_matrix( om, 1,NPK,1,2 ); om = NULL;
::free_ivector( bandcode, 1,nWvf_->scanl() ); bandcode = NULL;
 NPK = jdx-1;
refPks.set( nband_, NPK );
return setBandStats( refPks, FBW, seq_, pass2 );
}
```

```
/*************************************************************************
 * FILE:        nrutil.cxx
 * TYPIST:      Andy Marks
 */
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <nrc/nrutil.hxx>
#define NR_END 1
#define FREE_ARG char*
void
nrerror( char const error_text[] )
{
   fprintf(stderr,"Numerical Recipes run-time error..\n");
   fprintf(stderr,"%s\n", error_text );
   fprintf(stderr,"...now exiting to system...\n");
   exit(1);
}
float*
vector( long nl, long nh )
{
 float *v;
 v = (float*)malloc((size_t)((nh-nl+1+NR_END)*sizeof(float)));
 if(NULL == v) nrerror("allocation failure in vector()");
 return v-nl+NR_END;
}
int*
ivector( long nl, long nh )
{
 int *v;
 v = (int*)malloc((size_t)((nh-nl+1+NR_END)*sizeof(int)));
```

```
 if(NULL == v) {
  ::fprintf(stderr,"ivector: nl=%ld nh=%ld v=%p\n",nl,nh,v);
  ::nrerror("ivector: allocation failure");
  }
 return v-nl+NR_END;
}
unsigned char*
cvector( long nl, long nh )
{
 unsigned char *v;
 v = (unsigned char*)malloc((size_t)((nh-nl+1+NR_END)*sizeof(unsigned char)));
 if(NULL == v) nrerror("allocation failure in cvector()");
 return v-nl+NR_END;
}
unsigned long*
lvector( long nl, long nh )
{
 unsigned long *v;
 v = (unsigned long*)malloc((size_t)((nh-nl+1+NR_END)*sizeof(unsigned long)));
 if(NULL == v) nrerror("allocation failure in lvector()");
 return v-nl+NR_END;
}
double*
dvector( long nl, long nh )
{
 double *v;
 v = (double*)malloc((size_t)((nh-nl+1+NR_END)*sizeof(double)));
 if(NULL == v) nrerror("allocation failure in dvector()");
 return v-nl+NR_END;
}
float**
```

```
matrix(long nrl,long nrh,long ncl,long nch)
{
 long i, nrow = nrh-nrl+1, ncol=nch-ncl+1;
 float **m;
 m = (float**)malloc((size_t)((nrow+NR_END)*sizeof(float*)));
 if(NULL == m) nrerror("allocation failure 1 in matrix()");
 m += NR_END;
 m -= nrl;
 m[nrl] = (float*)malloc((size_t)((nrow*ncol+NR_END)*sizeof(float)));
 if(NULL == m[nrl]) nrerror("allocation failure 2 in matrix()");
 m[nrl] += NR_END;
 m[nrl] -= ncl;
 for(i=nrl+1;i<=nrh;i++) m[i] = m[i-1]+ncol;
 return m;
}
double**
dmatrix(long nrl,long nrh,long ncl,long nch)
{
 long i, nrow = nrh-nrl+1, ncol=nch-ncl+1;
 double **m;
 m = (double**)malloc((size_t)((nrow+NR_END)*sizeof(double*)));
 if(NULL == m) {
  ::fprintf(stderr,"nrl=%ld, nrh=%ld, ncl=%ld, nch=%ld, nrow=%ld\n",
   nrl,nrh,ncl,nch,nrow);
  nrerror("allocation failure 1 in dmatrix()");
  }
 m += NR_END;
 m -= nrl;
 m[nrl] = (double*)malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
 if(NULL == m[nrl]) nrerror("allocation failure 2 in dmatrix()");
 m[nrl] += NR_END;
```

```
 m[nrl] -= ncl;
 for(i=nrl+1;i<=nrh;i++) m[i] = m[i-1]+ncol;
 return m;
}
int**
imatrix(long nrl,long nrh,long ncl,long nch)
{
 long i, nrow = nrh-nrl+1, ncol=nch-ncl+1;
 int **m;
 m = (int**)malloc((size_t)((nrow+NR_END)*sizeof(int*)));
 if(NULL == m) nrerror("allocation failure 1 in dmatrix()");
 m += NR_END;
 m -= nrl;
 m[nrl] = (int*)malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
 if(NULL == m[nrl]) nrerror("allocation failure 2 in imatrix()");
 m[nrl] += NR_END;
 m[nrl] -= ncl;
 for(i=nrl+1;i<=nrh;i++) m[i] = m[i-1]+ncol;
 return m;
}
float**
submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,
 long newrl, long newcl)
{
 long i,j,nrow=oldrh-oldrl+1,ncol=oldch-oldcl+1;
 float **m;
 m = (float**)malloc((size_t)((nrow+NR_END)*sizeof(float*)));
 if(NULL == m) nrerror("allocation failure in submatrix()");
 m += NR_END;
 m -= newrl;
 for(i = oldrl, j=newrl; i<=oldrh; i++,j++) m[j] = a[i]+ncol;
```

```
 return m;
}
float**
convert_matrix(float* a, long nrl, long nrh, long ncl, long nch)
{
 long i, j, nrow=nrh-nrl+1, ncol=nch-ncl+1;
 float **m;
 m = (float**)malloc((size_t)((nrow+NR_END)*sizeof(float*)));
 if(NULL == m) nrerror("allocation failure in convert_matrix()");
 m += NR_END;
 m -= nrl;
 m[nrl] = a-ncl;
 for(i=1,j=nrl+1; i<=nrow; i++,j++) m[j] = m[j-1]+ncol;
 return m;
}
void
free_vector(float* v, long nl, long nh)
{
 free((FREE_ARG)(v+nl-NR_END));
}
void
free_ivector(int* v, long nl, long nh)
{
 free((FREE_ARG)(v+nl-NR_END));
}
void
free_cvector(unsigned char* v, long nl, long nh)
{
 free((FREE_ARG)(v+nl-NR_END));
}
void
```

```
free_lvector(unsigned long* v, long nl, long nh)
{
 free((FREE_ARG)(v+nl-NR_END));
}
void
free_dvector(double* v, long nl, long nh)
{
 free((FREE_ARG)(v+nl-NR_END));
}
void
free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
{
 free((FREE_ARG)(m[nrl]+ncl-NR_END));
 free((FREE_ARG)(m+nrl-NR_END));
}
void
free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)
{
 free((FREE_ARG)(m[nrl]+ncl-NR_END));
 free((FREE_ARG)(m+nrl-NR_END));
}
void
free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
{
 free((FREE_ARG)(m[nrl]+ncl-NR_END));
 free((FREE_ARG)(m+nrl-NR_END));
}
void
free_submatrix(int **b, long nrl, long nrh, long ncl, long nch)
{
 free((FREE_ARG)(b+nrl-NR_END));
```

```
}
void
free_convert_matrix(int  *b, long nrl)
{
 free((FREE_ARG)(b+nrl-NR_END));
}

/*****************************************************************************
 * FILE:        Pkdet.cxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
PKDET::PKDET() :
 status_(STS_INITD), ppk_(NULL), ptr_(NULL), gap_(NULL),
 wid_(NULL), ins_(NULL), Np_(0), Nt_(0), medGap_(0), medWid_(0)
{
}
PKDET::PKDET( PKDET const& rhs ) :
 status_(STS_INITD), ppk_(NULL), ptr_(NULL), gap_(NULL),
 wid_(NULL), ins_(NULL), Np_(0), Nt_(0), medGap_(0), medWid_(0)
{
  *this = rhs;
}
PKDET const&
PKDET::operator=(PKDET const& rhs)
{
  if(&rhs != this) {
   release();
   Np_ = rhs.Np_;
   Nt_ = rhs.Nt_;
```

```
        medGap_ = rhs.medGap_;
        medWid_ = rhs.medWid_;
        ppk_ = ::ivector( 1, Np_ );
        wid_ = ::ivector( 1, Np_ );
        ins_ = ::ivector( 1, Np_ );
        ptr_ = ::ivector( 1, Nt_ );
        gap_ = ::ivector( 1, Nt_ );
        if(!ppk_ || !ptr_ || !gap_ || !wid_ || !ins_) {
         status_ = STS_NO_MEM;
         release();
         }
        else {
         for(int idx = 1; idx<=Np_; idx++) {
          ppk_[ idx ] = rhs.ppk_[ idx ];
          wid_[ idx ] = rhs.wid_[ idx ];
          ptr_[ idx ] = rhs.ptr_[ idx ];
          gap_[ idx ] = rhs.gap_[ idx ];
          ins_[ idx ] = rhs.ins_[ idx ];
          }
         ptr_[ idx ] = rhs.ptr_[ idx ];
         gap_[ idx ] = rhs.gap_[ idx ];
         }
        }
      return *this;
}
void
PKDET::release()
{
  if(NULL != ppk_) { ::free_ivector( ppk_, 1, Np_ ); ppk_ = NULL; }
  if(NULL != wid_) { ::free_ivector( wid_, 1, Np_ ); wid_ = NULL; }
  if(NULL != ins_) { ::free_ivector( ins_, 1, Np_ ); ins_ = NULL; }
```

```
    if(NULL != ptr_) { ::free_ivector( ptr_, 1, Nt_ ); ptr_ = NULL; }
    if(NULL != gap_) { ::free_ivector( gap_, 1, Nt_ ); gap_ = NULL; }
    Np_ = Nt_ = 0;
  }
PKDET::~PKDET()
{
  release();
}
static int
i_cmp(void const* e1, void const* e2)
{
  return (*(int const*)e1)-(*(int const*)e2);
}
int
PKDET::set( int const* Pp, int Np, int const* Pt, int Nt )
{
  int P1,T1, px1,pxn, tx1,txn, *gtmp, *wtmp;
  release();
  px1 = Pp[ P1=1 ]; pxn = Pp[ Np ];
  tx1 = Pt[ T1=1 ]; txn = Pt[ Nt ];
  if(px1 < tx1) P1++;
  if(pxn > txn) Np--;
  Np_ = Np-P1+1;
  Nt_ = Nt;
  if((Nt_ < 2) || (Nt_ != (Np_+1))) {
   ::fprintf(stderr,"PKDET::set, Np=%d, Nt=%d: Nt should == Np+1\n",Np_,Nt_);
   ::fprintf(stderr,"\tpx1=%d tx1=%d pxn=%d txn=%d\n",px1,tx1,pxn,txn);
   Nt_ = Np_ = 0;
   return 0;
   }
  ppk_ = ::ivector( 1, Np_ );
```

```
wid_ = ::ivector( 1, Np_ );
ins_ = ::ivector( 1, Np_ );
wtmp = ::ivector( 1, Np_ );
gtmp = ::ivector( 1, Np_-1 );
ptr_ = ::ivector( 1, Nt_ );
gap_ = ::ivector( 1, Nt_ );
if(!ppk_ || !ptr_ || !gap_ || !wid_ || !ins_ || !gtmp || !wtmp) {
 status_ = STS_NO_MEM;
 Nt_ = Np_ = 0;
 return 0;
 }
for(int idx=1; idx<=Np_; idx++) {
 ppk_[ idx ] = Pp[ P1++ ];
 ptr_[ idx ] = Pt[ T1++ ];
 wid_[ idx ] = Pt[ T1 ] - Pt[ T1-1 ];
 ins_[ idx ] = 0;
 wtmp[ idx ] = wid_[ idx ];
 if(idx < Np_){
  gap_[ 1+idx ] = Pp[ P1 ] - Pp[ P1-1 ];
  gtmp[ idx ]  = gap_[ 1+idx ];
  }
 }
ptr_[idx] = Pt[T1]:
::qsort( &gtmp[1], Np_-1, sizeof(*gtmp), i_cmp );
if((Np_-1)&1) {
 int mid = 1+(Np_-1)/2;
 medGap_ = gtmp[mid];
 }
else {
 int mid = (Np_-1)/2;
 medGap_ = (gtmp[mid] + gtmp[mid+1])/2;
```

```
    }
  gap_[1] = gap_[Nt_] = medGap_;
  ::free_ivector( gtmp, 1,Np_-1 );
  ::qsort( &wtmp[1], Np_, sizeof(*wtmp), i_cmp );
  medWid_ = (Np_&1)? wtmp[1+Np_/2]: (wtmp[Np_/2]+wtmp[1+Np_/2])/2;
  ::free_ivector( wtmp, 1,Np_ );
  return 1;
}
int
PKDET::set( Band* Pb, int Nb )
{
  int idx, *gtmp, *wtmp;
  release();
  Np_ = Nb;
  Nt_ = Nb+1;
  if(Nt_ < 2) {
   ::fprintf(stderr,"PKDET::set, Np=%d, Nt=%d: Nt!=Np+1\n",Np_,Nt_);
   Nt_ = Np_ = 0;
   return 0;
   }
  ppk_ = ::ivector( 1, Np_ );
  wid_ = ::ivector( 1, Np_ );
  ins_ = ::ivector( 1, Np_ );
  ptr_ = ::ivector( 1, Nt_ );
  gap_ = ::ivector( 1, Nt_ );
  wtmp = ::ivector( 1, Np_ );
  gtmp = ::ivector( 1, Np_-1 );
  if(!ppk_ || !ptr_ || !gap_ || !wid_ || !ins_ || !gtmp || !wtmp) {
   status_ = STS_NO_MEM;
   Nt_ = Np_ = 0;
   return 0;
```

```
  }
 for(idx=1; idx<Nb; idx++) {
  Band bn  = Pb[idx], bnp1 = Pb[idx+1];
  if(bn.end() != bnp1.bgn())
   if(bn.wid() > bnp1.wid())
    Pb[idx].end( bnp1.bgn() );
   else
    Pb[idx+1].bgn( bn.end() );
  ppk_[idx]  = Pb[idx].mid();
  ptr_[idx]  = Pb[idx].bgn();
  wid_[idx]  = Pb[idx].wid();
  ins_[idx]  = Pb[idx].ins();
  wtmp[idx]  = wid_[idx];
  gap_[1+idx] = Pb[idx+1].mid() - Pb[idx].mid();
  gtmp[idx]  = gap_[1+idx];
  }
 ppk_[ idx ]  = Pb[ idx ].mid();
 ptr_[ idx ]  = Pb[ idx ].bgn();
 ptr_[ idx+1 ] = Pb[ idx ].end();
 wid_[ idx ]  = ptr_[idx+1] - ptr_[idx];
 ins_[ idx ]  = Pb[ idx ].ins();
 wtmp[ idx ]  = wid_[ idx ];
 ::qsort( &gtmp[1], Np_-1, sizeof(*gtmp), i_cmp );
 if((Np_-1)&1) {
  int mid = 1+(Np_-1)/2;
  medGap_ = gtmp[mid];
  }
 else {
  int mid = (Np_-1)/2;
  medGap_ = (gtmp[mid]+gtmp[mid+1])/2;
  }
```

```
    gap_[1] = gap_[Nt_] = medGap_;
    ::free_ivector( gtmp, 1,Np_-1 );
    ::qsort( &wtmp[1], Np_, sizeof(*wtmp), i_cmp );
    medWid_ = (Np_&1)? wtmp[1+Np_/2]: (wtmp[Np_/2]+wtmp[1+Np_/2])/2;
    ::free_ivector( wtmp, 1,Np_ );
    return 1;
}
void
PKDET::debug( int lvl ) const
{
    ::printf("PKDET::debug\n");
    ::printf("Np=%3d  Nt=%3d  medGap=%2d  medWid=%2d\n",
     Np_,Nt_,medGap_,medWid_);
    if(NULL == ptr_)
     ::printf("PKDET::debug failed: ptr_=%p\n",ptr_);
    else if(NULL == ppk_)
     ::printf("PKDET::debug failed: ppk_=%p\n",ppk_);
    else if(NULL == gap_)
     ::printf("PKDET::debug failed: gap_=%p\n",gap_);
    else if(NULL == wid_)
     ::printf("PKDET::debug failed: wid_=%p\n",wid_);
    else if(NULL == ins_)
     ::printf("PKDET::debug failed: ins_=%p\n",ins_);
    else for(int idx=1; idx<=Np_; idx++) {
     ::printf("%3d: %4d|%4d|%4d, lgap=%2d rgap=%2d width=%2d ins=%d\n",
      idx,ptr_[idx],ppk_[idx],ptr_[idx+1],gap_[idx],gap_[idx+1],
      wid_[idx],ins_[idx]);
     ::fflush( stdout );
     }
}
```

```
/*************************************************************************
 * FILE:        Pkdet.hxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#if !defined(_PKDET_HXX_)
# define _PKDET_HXX_
#include <stdio.h>
class Band
{
public:
  Band() : b_(0), m_(0), e_(0), ins_(0) {;}
  Band(int b, int m, int e, int i) : b_(b), m_(m), e_(e), ins_(i) {;}
  ~Band() {;}
  int bgn() const { return b_; }
  int mid() const { return m_; }
  int end() const { return e_; }
  int wid() const { return e_-b_+1; }
  int ins() const { return ins_; }
  void bgn( int b ) { b_ = b; }
  void mid( int m ) { m_ = m; }
  void end( int e ) { e_ = e; }
  void ins( int i ) { ins_ = i; }
  void debug( int lvl=0 ) const
  {
    ::printf("%d,%d,%d (inserted=%d)\n",b_,m_,e_,ins_);
    ::fflush(stdout);
  }
private:
  short b_, m_, e_, ins_;
};
```

```
class PKDET
{
public:
  enum Status { STS_UNINITD, STS_INITD, STS_NO_MEM };
  PKDET();
  PKDET(PKDET const& rhs);
  PKDET const& operator=(PKDET const& rhs);
  ~PKDET();
  int set( int const* ppk, int Np, int const* ptr, int Nt );
  int set( Band* pb, int Nb );
  Status status()   const { return status_; }
  int npk()         const { return Np_; }
  int ntr()         const { return Nt_; }
  int const* bbgn()  const { return ptr_; }
  int const* bmid()  const { return ppk_; }
  int const* bend()  const { return &ptr_[1]; }
  int ins(int idx)   const { return ins_[idx]; }
  int const* bwid()  const { return wid_; }
  int bbgn(int idx)  const { return ptr_[idx]; }
  int bmid(int idx)  const { return ppk_[idx]; }
  int bend(int idx)  const { return ptr_[1+idx]; }
  int bwid(int idx)  const { return wid_[idx]; }
  int lgap(int idx)  const { return gap_[idx]; }
  int rgap(int idx)  const { return gap_[1+idx]; }
  Band band(int idx) const {
   Band b(bbgn(idx),bmid(idx),bend(idx),ins(idx));
   return b;
  }
  int const* lgap()  const { return gap_; }
  int const* rgap()  const { return &gap_[1]; }
  int medGap()       const { return medGap_; }
```

```
  int medWid()      const { return medWid_; }
  void debug( int lvl = 0 ) const;
private:
  Status status_;
  int *ppk_,
    *ptr_,
    *gap_,
    *wid_,
    *ins_;
  int  Np_,
     Nt_,
     medGap_,
     medWid_;
  void release();
};
#endif

/*****************************************************************************
 * FILE:        polfit.cxx
 * AUTHOR:  Philip R. Bevington, p140-141
 */
#include <nrc/nr.hxx>
#if !defined(SA)
int
polfit(  float const* px, float const* py,
         float const* sigmaY,
         int NPTS,
         int NTERMS,
         PFWGHT mode,
         float coef[],
         float& chisq )
```

```
{
  int NMAX, inr, jnr;
  float degfre, **soln, *sumx, *sumy, **array;
  NMAX = 2*NTERMS-1;
  sumx  = ::vector(1,NMAX);
  sumy  = ::vector(1,NTERMS);
  soln  = ::matrix(1,NTERMS,1,1);
  array = ::matrix(1,NTERMS,1,NTERMS);
  for(inr=1;inr<=NMAX;inr++)
   sumx[inr] = 0.0f;
  for(inr=1;inr<=NTERMS;inr++)
   sumy[inr] = 0.0f;
  chisq = 0.0f;
  for(inr=1;inr<=NPTS;inr++) {
   float x, y, wt, xterm, yterm;
   x = px[inr];
   y = py[inr];
   switch(mode) {
    case INSTRUMENTAL:
     if(y<0)  wt = -1.0f/y;
     else if(0==y) wt = 1.0f;
     else          wt = 1.0f/y;
     break;
    case NO_WEIGHTING:
     wt = 1.0f;
     break;
    case STATISTICAL:
     wt = 1.0f/(sigmaY[inr]*sigmaY[inr]);
     break;
   }
  xterm = wt;
```

```
    for(jnr=1;jnr<=NMAX;jnr++) {
     sumx[jnr] += xterm;
     xterm *= x;
     }
    yterm = wt*y;
    for(jnr=1;jnr<=NTERMS;jnr++) {
     sumy[jnr] += yterm;
     yterm *= x;
     }
    chisq += wt*y*y;
   }
  for(inr=1;inr<=NTERMS;inr++) {
   soln[inr][1] = sumy[inr];
   for(jnr=1;jnr<=NTERMS;jnr++)
    array[inr][jnr] = sumx[inr+jnr-1];
   }
  ::gaussj( array,NTERMS, soln,1);
  for(inr=1;inr<=NTERMS;inr++) {
   coef[inr] = soln[inr][1];
   chisq -= 2.0f*soln[inr][1]*sumy[inr];
   for(jnr=1;jnr<=NTERMS;jnr++)
    chisq +=  soln[inr][1]*soln[jnr][1]*sumx[inr+jnr-1];
   }
  ::free_vector(sumx,1,NMAX);
  ::free_vector(sumy,1,NTERMS);
  ::free_matrix(soln,1,NTERMS,1,1);
  ::free_matrix(array,1,NTERMS,1,NTERMS);
  if(0.0f == (degfre = float(NPTS - NTERMS)))
   return 1;
  chisq /= degfre;
  return 0;
```

```
}
#endif
#if defined(SA)
#include <stdio.h>
int
main(int argc, char* argv[])
{
  static float x[] =
      {
        0.0f,
     282.0f, 338.0f, 393.0f, 460.0f, 486.0f,
     512.0f, 564.0f, 615.0f, 668.0f, 726.0f,
       780.0f, 832.0f, 886.0f, 940.0f, 993.0f,
    1047.0f,1099.0f,1152.0f,1204.0f,1262.0f,
       1317.0f,1370.0f,1424.0f,1476.0f,1532.0f,
    1587.0f,1640.0f,1694.0f,1746.0f,1801.0f
      };
  static float y[] =
      {
      0.0f,
       1436.0f,1420.0f,1408.0f,1404.0f,1400.0f,
    1388.0f,1384.0f,1364.0f,1364.0f,1352.0f,
       1348.0f,1340.0f,1340.0f,1336.0f,1328.0f,
    1328.0f,1328.0f,1328.0f,1324.0f,1328.0f,
       1328.0f,1324.0f,1324.0f,1324.0f,1324.0f,
    1320.0f,1332.0f,1332.0f,1332.0f,1328.0f
      };
# define NPTS (sizeof(x)/sizeof(x[0]) - 1)
  float ochisq;
  for(int tnr=2;tnr<=6;tnr++) {
   float chisq, coef[8];
```

```
    (void)polfit(x,y,NULL,NPTS,tnr,NO_WEIGHTING,coef,chisq);
    ::printf("\ntnr=%d chisq=%f\n",tnr,chisq);
    for(int t=1;t<=tnr;t++)
     ::printf("\tcoef[%d]=%f\n",t,coef[t]);
    if(2!=tnr) {
     float improv = 100.0f*ochisq/chisq;
     ::printf("IMPROVED BY %7.2f%%\n",improv);
     if(improv < 115.0f)
      break;
     }
    ochisq = chisq;
    }
  }
  #endif

  /*************************************************************************
   * FILE:        preproc.cxx
   * AUTHOR:  Andy Marks
   * COPYRIGHT (c) 1996, University of Utah
   */
  #include <basecall/mb.hxx>
  struct MINV { double v; int i; };
  void
  Wvfm::truvelAdjust()
  {
    double const EXPONENT = 1.0/2.2;
    double CONSTANT = ::pow( 1.0/255.0, EXPONENT );
    double minv[5];
    int ldx, sdx;
    for(ldx = 1; ldx <= lanes(); ldx++) {
     minv[ldx] = 1000.0;
```

```
        for(sdx = bgni(); sdx <= endi(); sdx++) {
         double v = sc_la(sdx,ldx);
         if(v <= 63.0)      v = v*0.2511;
         else if(v <= 127.0) v = (v*0.7028 - 255.0*0.1129);
         else if(v <= 191.0) v = (v*1.245  - 255.0*0.3887);
         else                v = (v*1.7916 - 255.0*0.7916);
         v = pow( 1.0/v, EXPONENT ) - CONSTANT;
         sc_la_set( sdx, ldx, v );
         if(v < minv[ldx]) minv[ldx] = v;
         }
        }
        for(ldx = 1; ldx <= lanes(); ldx++) {
         double mn = minv[ldx];
         for(sdx = bgni(); sdx <= endi(); sdx++)
          sc_la_sub( sdx,ldx, mn );
         }
       }
       static void
       minimum( double **pm, int ROW, int COL, int N, MINV& rm )
       {
        rm.v = pm[ rm.i=ROW ][ COL ];
        for(int sdx = ROW+1; sdx<ROW+N; sdx++)
         if( pm[ sdx ][ COL ] < rm.v )
          rm.v = pm[ rm.i=sdx ][ COL ];
       }
       void
       Wvfm::baseline( int N )
       {
          int LEN = endi()-bgni()+1, sdx, ldx;
          MINV minv[5];
          double **pm;
```

```
    pm = ::dmatrix( 1, LEN + (2*N), 1, lanes() );
    for(ldx=1; ldx<=lanes(); ldx++) {
     double S1 = sc_la( bgni(), ldx ),
          SN = sc_la( endi(), ldx );
     for(sdx=1; sdx<=N; sdx++) {
      pm[sdx][      ldx ] = S1;
      pm[N+LEN+sdx][ ldx ] = SN;
      }
     for(sdx=1; sdx<=LEN; sdx++)
      pm[N+sdx ][ ldx ] = sc_la( bgni()+sdx-1, ldx);
     }
    for(ldx=1; ldx<=lanes(); ldx++)
     minimum( pm, 1, ldx, 2*N, minv[ldx] );
    for(ldx = 1; ldx <= lanes(); ldx++) {
     int irnr, lrnr;
     irnr = bgni();
     for(lrnr = N+1; lrnr <= N+LEN; irnr++) {
      pm_[ irnr ][ ldx ] = pm[lrnr++][ldx] - minv[ldx].v;
      if( minv[ ldx ].i < (lrnr-N) )
       minimum( pm, lrnr-N, ldx, 2*N, minv[ldx] );
      else if( minv[ ldx ].v > pm[ lrnr+N-1 ][ ldx ])
       minv[ ldx ].v = pm[ minv[ ldx ].i = lrnr+N-1 ][ ldx ];
      }
     }
    ::free_dmatrix(pm,1,LEN+2*N, 1, lanes());
}
void
Wvfm::noZeros()
{
   for(int ldx = 1; ldx <= lanes(); ldx++)
    for(int sdx = bgni(); sdx <= endi(); sdx++)
```

```
        if(0.0 == pm_[sdx][ldx])
          pm_[sdx][ldx] = DBL_EPSILON;
    }
    int
    Wvfm::mdynpre()
    {
      bgnEnd();
      if(1 != specSep())
        return 0;
      if(_1X != ib_) {
        double **pn;
        float *x, *y, *y2;
        int r,R,s,c,k,NF,N,NP;
        NP = (_3X==ib_)? 3: 2;
        NF = (scanl()+1)/NP;
        N  = (NF*NP*NP+1);
        if(NULL == (pn = ::dmatrix( 1,N, 1,lanes()))) {
          status_ = STS_NO_MEM;
          return 0;
          }
        x = ::vector(1,4);
        y = ::vector(1,4);
        y2 = ::vector(1,4);
        if(!x || !y || !y2) {
          status_ = STS_NO_MEM;
          return 0;
          }
        for(c=1; c<=lanes(); c++) {
          R = (bgni()-1)*NP+1;
          for(r=bgni(); r<=(endi()-3); r+=NP) {
            for(s=0; s<=3; s++) {
```

```
            x[s+1] = float(r+s);
            y[s+1] = float(sc_la(r+s,c));
            }
          ::spline( x, y, 4, y[2]-y[1], y[4]-y[3], y2);
          for(s=0; s<NP; s++)
            for(k=0; k<NP; k++) {
              float yout, xin;
              xin = float(r+s)+float(k)/float(NP);
              ::splint( x, y, y2, 4, xin, &yout);
              pn[R++][c] = double(yout);
              }
            }
          pn[R][c] = y[4];
          }
        ::free_vector(x,1,4);
        ::free_vector(y,1,4);
        ::free_vector(y2,1,4);
        pm( pn, R, bgni()*NP-NP+1,endi()*NP-NP+1 );
        }
      return 1;
    }
    int
    Wvfm::preproc()
    {
      if(MDYN == ds_)
        if(1 != mdynpre())
          return 0;
      if(TRUVEL == ds_)
        truvelAdjust();
      else if(MDYN != ds_)
        baseline( 50 );
```

```
    noZeros();
    bgni_++;
    endi_--;
    return 1;
}

/*****************************************************************
 * FILE:        Quadratic.cxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <stdio.h>
#include <float.h>
#if defined(sun)
# include <ieeefp.h>
#endif
#if defined(_WIN32)
# define finite _finite
#endif
#include <nrc/nr.hxx>
int
ilinreg( int const* px, int const* py, int N, float* coef, float* prr)
{
  float **m, **v, muY=0.0f;
  double sx, sx2, ss_about_mu=0.0, ss_due_reg=0.0, ss_about_reg=0.0;
  int idx;
  coef[0] = coef[1] = coef[2] = 0.0f;
  if(N<2) return 0;
  if(NULL == (m = ::matrix(1,2,1,2)))
   return 0;
  if(NULL == (v = ::matrix(1,2,1,1))) {
```

```
    if(m) ::free_matrix(m,1,2,1,2);
    return 0;
    }
   v[1][1] = v[2][1] = 0.0f;
   sx = sx2 = 0.0;
   for(idx=1; idx<=N; idx++) {
    double x = double(px[idx]), y = double(py[idx]);
    sx += x; sx2 += x*x;
    v[1][1] += float(y);
    v[2][1] += float(x*y);
    }
  muY = v[1][1]/float(N);
  m[1][1] = float(N);
  m[1][2] = m[2][1] = float(sx);
  m[2][2] = float(sx2);
  gaussj( m,2, v,1 );
  coef[0] = v[1][1];
  coef[1] = v[2][1];
  for(idx=1;idx<=N;idx++) {
   double yh = coef[1]*double(px[idx]) + coef[0];
   double y  = double(py[idx]);
   ss_about_mu  += (y-muY)*(y-muY);
   ss_due_reg   += (yh-muY)*(yh-muY);
   ss_about_reg += (y-yh)*(y-yh);
   }
 if(N>2)
  coef[2] = float(ss_about_reg/double(N-2));
 if(NULL != prr)
  *prr = float(ss_due_reg/ss_about_mu);
 ::free_matrix(m,1,2,1,2);
 ::free_matrix(v,1,2,1,1);
```

```
    return 1;
}
int
linreg( float const* px, float const* py, int N, float* coef, float* prr)
{
    float **m, **v, muY=0.0f;
    double sx, sx2, ss_about_mu=0.0, ss_due_reg=0.0, ss_about_reg=0.0;
    int idx;
    coef[0] = coef[1] = coef[2] = 0.0f;
    if(N<2) return 0;
    if(NULL == (m = ::matrix(1,2,1,2)))
     return 0;
    if(NULL == (v = ::matrix(1,2,1,1))) {
     if(m) ::free_matrix(m,1,2,1,2);
     return 0;
     }
    v[1][1] = v[2][1] = 0.0f;
    sx = sx2 = 0.0;
    for(idx=1; idx<=N; idx++) {
     float x=px[idx], y=py[idx];
     sx += x; sx2 += x*x;
     v[1][1] += float(y);
     v[2][1] += float(x*y);
     }
    muY = v[1][1]/float(N);
    m[1][1] = float(N);
    m[1][2] = m[2][1] = float(sx);
    m[2][2] = float(sx2);
    gaussj( m,2, v,1 );
    coef[0] = v[1][1];
    coef[1] = v[2][1];
```

```
    for(idx=1;idx<=N;idx++) {
      double yh = coef[1]*double(px[idx]) + coef[0];
      double y  = double(py[idx]);
      ss_about_mu  += (y-muY)*(y-muY);
      ss_due_reg   += (yh-muY)*(yh-muY);
      ss_about_reg += (y-yh)*(y-yh);
      }
    if(N>2)
      coef[2] = float(ss_about_reg/double(N-2));
    if(NULL != prr)
      *prr = float(ss_due_reg/ss_about_mu);
    ::free_matrix(m,1,2,1,2);
    ::free_matrix(v,1,2,1,1);
    return 1;
}
int
iquadratic( int const* px, int const* py, int N, float* quad)
{
    float **m, **v;
    double sx, sx2, sx3, sx4, uy;
    double sum;
    int idx;
    quad[0] = quad[1] = quad[2] = quad[3] = 0.0f;
    if(N<=3) return 0;
    m = ::matrix(1,3,1,3);
    if(!m)
      return 0;
    v = ::matrix(1,3,1,1);
    if(!v) {
      ::free_matrix(m,1,3,1,3);
      return 0;
```

```
    }
  v[1][1] = v[2][1] = v[3][1] = 0.0f;
  sx = sx2 = sx3 = sx4 = 0.0;
  for(idx=1; idx<=N; idx++) {
   double x = double(px[idx]), y = double(py[idx]);
   sx += x; sx2 += x*x;  sx3 += x*x*x; sx4 += x*x*x*x;
   v[1][1] += float(y);
   v[2][1] += float(x*y);
   v[3][1] += float(x*x*y);
   }
  uy = v[1][1]/float(N);
  sum = 0.0;
  for(idx=1; idx<=N; idx++) {
   double y = double(py[idx]);
   sum += (y-uy)*(y-uy);
   }
  sum /= double(N-1);
  if(0.0 == sum)
   quad[0] = float(uy);
  else {
   v[1][1] /= float(sum);
        v[2][1] /= float(sum);
        v[3][1] /= float(sum);
   m[1][1] = float(float(N)/sum);
   m[1][2] = m[2][1] = float(sx/sum);
   m[1][3] = m[2][2] = m[3][1] = float(sx2/sum);
   m[2][3] = m[3][2] = float(sx3/sum);
   m[3][3] = float(sx4/sum);
   gaussj( m,3, v,1);
   sum = 0.0;
   for(idx=1; idx<=N; idx++) {
```

```
        double x = double(px[idx]), y = double(py[idx]), yprim;
        yprim = v[1][1] + v[2][1]*x + v[3][1]*x*x;
        sum   += (yprim-y)*(yprim-y);
        }
      quad[0] = v[1][1];
      quad[1] = v[2][1];
      quad[2] = v[3][1];
      quad[3] = float(sum/double(N-2-1));
      }
    ::free_matrix(m,1,3,1,3);
    ::free_matrix(v,1,3,1,1);
    return 1;
}
int
dquadratic( double const* px, double const* py, int N, float* coef)
{
    float **m, **v;
    double sx, sx2, sx3, sx4, uy, sum;
    int idx;
    coef[0] = coef[1] = coef[2] = coef[3] = 0.0f;
    if(N<=3) return 0;
    m = ::matrix(1,3,1,3);
    if(!m)
     return 0;
    v = ::matrix(1,3,1,1);
    if(!v) {
     ::free_matrix(m,1,3,1,3);
     return 0;
     }
    v[1][1] = v[2][1] = v[3][1] = 0.0f;
    sx = sx2 = sx3 = sx4 = 0.0;
```

```
for(idx=1; idx<=N; idx++) {
 double x=px[idx], y=py[idx];
 sx  += x;
 sx2 += x*x;
 sx3 += x*x*x;
 sx4 += x*x*x*x;
 v[1][1] += float( y);
 v[2][1] += float( x*y);
 v[3][1] += float( x*x*y);
 }
uy = v[1][1]/float(N);
sum = 0.0;
for(idx=1; idx<=N; idx++) {
 double y = double(py[idx]);
 sum += (y-uy)*(y-uy);
 }
sum /= double(N-1);
if(0.0 == sum) {
 coef[0] = float(uy);
      coef[1] = coef[2] = coef[3] = 0.0f;
 }
else {
 v[1][1] /= float(sum);
 v[2][1] /= float(sum);
 v[3][1] /= float(sum);
 m[1][1] = float(float(N)/sum);
 m[1][2] = m[2][1] = float(sx/sum);
 m[1][3] = m[2][2] = m[3][1] = float(sx2/sum);
 m[2][3] = m[3][2] = float(sx3/sum);
 m[3][3] = float(sx4/sum);
 gaussj( m,3, v,1);
```

```
    sum = 0.0;
    for(idx=1; idx<=N; idx++) {
     double x = px[idx], y = py[idx], yprim;
     yprim = v[1][1] + v[2][1]*x + v[3][1]*x*x;
     sum  += (yprim-y)*(yprim-y);
     }
    coef[0] = float(v[1][1]);
    coef[1] = float(v[2][1]);
    coef[2] = float(v[3][1]);
    coef[3] = float(sum/double(N-2-1));
    }
   ::free_matrix(m,1,3,1,3);
   ::free_matrix(v,1,3,1,1);
   if(!finite(coef[3])) coef[3] = float(1E6);
   return 1;
}

/*************************************************************************
 * FILE:        Quadratic.hxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#if !defined(_QUADRATIC_HXX_)
# define _QUADRATIC_HXX_
void iquadratic(int const* x, int const* y, int n, float coef[4]);
void dquadratic(double const* x, double const* y, int n, float coef[4]);
#endif

/* ******** *** ************ ****************************************
 * FILE:        RatioBin.hxx
 * AUTHOR:  Andy Marks
```

```
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <basecall/RatioBin.hxx>
#include <nrc/nr.hxx>
RatioPattern::RatioPattern(int nfluor) : nfluor_(nfluor), npatrn_(0), chmrdy_(0)
{
  int pdx, rdx;
  for(pdx=0;pdx<MAXPTRN;pdx++) {
   PATRN& rp = patrn_[pdx];
   rp.frstAtBand = rp.lastAtBand = 0;
   rp.nentry = rp.t3 = rp.t4 = 0;
   rp.r34    = 0.0;
   for(rdx=0;rdx<nfluor_;rdx++)
    rp.rnsum[rdx] = rp.ratio[rdx] = 0.0f;
   }
#if 1
  PATRN* pp;
  pp = &patrn_[0]; pp->nentry=1; pp->t3=3; pp->t4=1; pp->r34=0.8f;
  pp->rnsum[0] = 1.0f; pp->rnsum[1] = 0.5f; pp->rnsum[2] = 0.5f; pp->rnsum[3] = 0.8f;
  pp->ratio[0] = 1.0f; pp->ratio[1] = 0.5f; pp->ratio[2] = 0.5f; pp->ratio[3] = 0.8f;
  pp = &patrn_[1]; pp->nentry=1; pp->t3=3; pp->t4=1; pp->r34=0.6f;
  pp->rnsum[0] = 0.4f; pp->rnsum[1] = 1.0f; pp->rnsum[2] = 0.3f; pp->rnsum[3] = 0.3f;
  pp->ratio[0] = 0.4f; pp->ratio[1] = 1.0f; pp->ratio[2] = 0.3f; pp->ratio[3] = 0.3f;
  pp = &patrn_[2]; pp->nentry=1; pp->t3=1; pp->t4=2; pp->r34=0.3f;
  pp->rnsum[0] = 0.0f; pp->rnsum[1] = 0.3f; pp->rnsum[2] = 1.0f; pp->rnsum[3] = 0.1f;
  pp->ratio[0] = 0.0f; pp->ratio[1] = 0.3f; pp->ratio[2] = 1.0f; pp->ratio[3] = 0.1f;
  pp = &patrn_[3]; pp->nentry=1; pp->t3=3; pp->t4=2; pp->r34=0.8f;
  pp->rnsum[0] = 0.3f; pp->rnsum[1] = 0.4f; pp->rnsum[2] = 1.0f; pp->rnsum[3] = 0.8f;
  pp->ratio[0] = 0.3f; pp->ratio[1] = 0.4f; pp->ratio[2] = 1.0f; pp->ratio[3] = 0.8f;
```

```
    npatrn_ = 4;
#endif
    for(pdx=0;pdx<nfluor_;pdx++)
     for(rdx=0;rdx<nfluor_;rdx++)
      chm_[pdx][rdx] = (pdx==rdx)?1.0:0.0;
}
void
RatioPattern::debug() const
{
  ::printf("P# #ent t3 t4 frst last rat34 ratfl1 ratfl2 ratfl3 ratfl4\n");
  for(int pdx=0;pdx<MAXPTRN;pdx++) {
   PATRN const& rp = patrn_[pdx];
   ::printf("%2d %4d %2d %2d %4d %4d %5.3f %6.3f %6.3f %6.3f %6.3f\n",
     pdx,rp.nentry,rp.t3,rp.t4,rp.frstAtBand,rp.lastAtBand,rp.r34,
     rp.ratio[0],rp.ratio[1],rp.ratio[2],rp.ratio[3]);
   ::printf("\n");
   }
}
int
RatioPattern::add( double const* obs, int bandnr )
{
  double mind=4.0f;
  int   minx=-1, pdx, rdx, rv=1;
  if(bandnr<=2 || npatrn_>=10) return 1;
  for(pdx=0;pdx<npatrn_;pdx++) {
   double eucd=0.0;
   for(rdx=0;rdx<nfluor_;rdx++) {
    double v = (patrn_[pdx].ratio[rdx]-obs[rdx]);
    eucd += v*v;
    }
   eucd = ::sqrt(eucd);
```

```
        if(eucd<mind) {
         mind=eucd;
         minx=pdx;
         }
       }
      if((-1==minx || mind>MAXEUCD) && npatrn_<MAXPTRN) {
       PATRN& rp = patrn_[minx = npatrn_++];
       rp.frstAtBand = bandnr;
       rp.nentry++;
       for(rdx=0;rdx<nfluor_;rdx++) {
        rp.ratio[rdx] = obs[rdx];
        rp.rnsum[rdx] = obs[rdx];
        }
       mind = 0.0;
       }
      else if(mind<=MAXEUCD) {
       PATRN& rp = patrn_[minx];
       rp.lastAtBand = bandnr;
       rp.nentry++;
       for(rdx=0;rdx<nfluor_;rdx++) {
        rp.ratio[rdx] = ((FLTRK-1.0)*rp.ratio[rdx] + obs[rdx])/FLTRK;
        rp.rnsum[rdx] += obs[rdx];
        }
       }
      else
       rv = 0;
     #if 0
       if(1==rv) {
        PATRN& rp = patrn_[minx];
        ::printf("OBS[nn](%5.2f %5.2f %5.2f %5.2f)|%5.3f|",
         obs[0],obs[1],obs[2],obs[3].mind);
```

```
    ::printf("PAT[%2d](%5.2f %5.2f %5.2f %5.2f)\n",
     minx,rp.ratio[0],rp.ratio[1],rp.ratio[2],rp.ratio[3]);
    }
#endif
  return rv;
}
void
RatioPattern::sortOnNentry_()
{
  for(int pdx=npatrn_-1;pdx>0;pdx--)
   for(int tdx=0;tdx<pdx;tdx++)
    if(patrn_[tdx].nentry < patrn_[tdx+1].nentry) {
     PATRN tmp = patrn_[tdx];
     patrn_[tdx] = patrn_[tdx+1];
     patrn_[tdx+1] = tmp;
     }
}
void
RatioPattern::sortOnR34_()
{
  int pdx, rdx, tdx;
  for(pdx=0;pdx<npatrn_;pdx++) {
   PATRN& rp = patrn_[pdx];
   double dtmp[4];
   int   itmp[4];
   for(rdx=0;rdx<nfluor_;rdx++) {
    rp.ratio[rdx] = rp.rnsum[rdx]/double(rp.nentry);
    dtmp[rdx] = rp.ratio[rdx];
    itmp[rdx] = rdx;
    }
   for(tdx=nfluor_-1;tdx>0;tdx--)
```

```
            for(rdx=0;rdx<tdx;rdx++)
             if(dtmp[rdx]>dtmp[rdx+1]) {
              double dt = dtmp[rdx]; dtmp[rdx] = dtmp[rdx+1];  dtmp[rdx+1] = dt;
              int    it = itmp[rdx]; itmp[rdx] = itmp[rdx+1]; itmp[rdx+1] = it;
              }
           rp.r34 = (0.0==dtmp[3])? 1.0: dtmp[2]/dtmp[3];
           rp.t4 = itmp[3];
           rp.t3 = itmp[2];
           }
         for(pdx=nfluor_-1;pdx>0;pdx--)
          for(tdx=0:tdx<pdx;tdx++)
           if(patrn_[tdx].r34 > patrn_[tdx+1].r34) {
            PATRN t = patrn_[tdx];
            patrn_[tdx] = patrn_[tdx+1];
            patrn_[tdx+1] = t;
            }
        }
        int
        RatioPattern::orderIt_()
        {
          PATRN* ordered = new PATRN[ nfluor_ ];
          int pdx, rv = 1;
          for(pdx=0;pdx<nfluor_;pdx++)
           ordered[pdx].nentry = 0;
          if(NULL == ordered)
           return 0;
          for(pdx=0;pdx<nfluor_;pdx++) {
           PATRN const& rp = patrn_[pdx];
           int mjr=rp.t4, mnr=rp.t3;
           if(0==ordered[mjr].nentry) ordered[mjr] = rp;
           else if(0==ordered[mnr].nentry) ordered[mnr] = rp;
```

```
        else {
         int alt = ordered[mjr].t3;
         if(0==ordered[alt].nentry) {
          ordered[alt] = ordered[mjr];
          ordered[mjr] = rp;
          }
         else {
          alt = ordered[mnr].t3;
          if(0==ordered[alt].nentry) {
           ordered[alt] = ordered[mnr];
           ordered[mnr] = rp;
           }
          else {
           rv = 0;
           goto bugout;
           }
          }
         }
        }
    for(pdx=0;pdx<nfluor_;pdx++)
     patrn_[pdx] = ordered[pdx];
bugout:
   delete [] ordered;
   return rv;
}
double
RatioPattern::chm(int rn, int cn)
{
   if(0==chmrdy_) {
    sortOnNentry_();
    sortOnR34_();
```

```
    if(1 == (chmrdy_ = orderIt_()))
     for(int pdx=0;pdx<nfluor_;pdx++)
      for(int tdx=0;tdx<nfluor_;tdx++)
       chm_[tdx][pdx] = patrn_[pdx].ratio[tdx];
   }
  return chm_[rn-1][cn-1];
}
RatioBin::RatioBin( int nf ) :  nfluor_(nf), CHM_(NULL), SST_(NULL), rp_(nf)
{
  if(nfluor_>1) {
   CHM_ = ::matrix(1,nfluor_,1,nfluor_);
   SST_ = ::matrix(1,nfluor_,1,nfluor_);
   }
}
RatioBin::RatioBin( RatioBin const& rhs ) :
 nfluor_(0), CHM_(NULL), SST_(NULL)
{
  *this = rhs;
}
void
RatioBin::release_()
{
  if(NULL != CHM_)  { ::free_matrix(CHM_,1,nfluor_,1,nfluor_); CHM_ = NULL; }
  if(NULL != SST_)  { ::free_matrix(SST_,1,nfluor_,1,nfluor_); SST_ = NULL; }
}
RatioBin const&
RatioBin::operator=( RatioBin const& rhs )
{
  if(&rhs != this) {
   release_();
   nfluor_ = rhs.nfluor_;
```

```
    rp_ = rhs.rp_;
    if(nfluor_>1) {
     if(NULL == (CHM_ = ::matrix(1,nfluor_,1,nfluor_)))
      goto bugout;
     if(NULL == (SST_ = ::matrix(1,nfluor_,1,nfluor_))) {
      ::free_matrix(CHM_,1,nfluor_,1,nfluor_);
      goto bugout;
      }
     for(int rdx=1;rdx<=nfluor_;rdx++)
      for(int cdx=1;cdx<=nfluor_;cdx++) {
       CHM_[rdx][cdx] = rhs.CHM_[rdx][cdx];
       SST_[rdx][cdx] = rhs.SST_[rdx][cdx];
       }
     }
    }
bugout:
  return *this;
}
RatioBin::~RatioBin()
{
  release_();
}
int
RatioBin::classify( double const * smpl, int x )
{
  double mxv, normalized[4];
  int idx, mxi;
  mxv = smpl[mxi=0];
  for(idx=1;idx<nfluor_;idx++)
   if(smpl[idx]>mxv)
    mxv = smpl[mxi=idx];
```

```
    for(idx=0;idx<nfluor_;idx++)
     normalized[idx] = smpl[idx]/mxv;
    (void)rp_.add( normalized, x );
    return 1;
}
void
RatioBin::debug(unsigned lvl) const
{
    int idx, cdx;
    ::printf("RatioBin @ %p\n",(void*)this);
    rp_.debug();
    ::printf("CHM @ %p:\n",(void*)CHM_);
    if(NULL != CHM_)
     for(idx=1;idx<=nfluor_;idx++) {
      ::printf("\t");
      for(cdx=1;cdx<=nfluor_;cdx++)
       ::printf("%8.5f ",CHM_[idx][cdx]);
      ::printf("\n");
      }
    ::printf("SST @ %p:\n",(void*)SST_);
    if(NULL != SST_)
     for(idx=1;idx<=nfluor_;idx++) {
      ::printf("\t");
      for(cdx=1;cdx<=nfluor_;cdx++)
       ::printf("%8.5f ",SST_[idx][cdx]);
      ::printf("\n");
      }
}
int
RatioBin::analyze()
{
```

```
float** CNT = ::matrix(1,nfluor_,1,1);
int idx, jdx;
for(idx=1;idx<=nfluor_;idx++) {
 CNT[idx][1] = 1.0f;
 for(jdx=1;jdx<=nfluor_;jdx++)
  CHM_[idx][jdx] = float(rp_.chm(idx,jdx));
 }
for(jdx=1;jdx<=nfluor_;jdx++)
 for(idx=1;idx<=nfluor_;idx++)
  SST_[idx][jdx] = CHM_[idx][jdx];
::gaussj( SST_, nfluor_, CNT, 1 );
#if 1
for(idx=1;idx<=nfluor_;idx++) {
 int mxi=1;
 float mxv=SST_[mxi][idx];
 for(jdx=2;jdx<=nfluor_;jdx++)
  if(SST_[jdx][idx]>mxv) {
   mxv = SST_[jdx][idx];
   mxi = jdx;
   }
 if(idx!=mxi) {
  ::fprintf(stderr,"fluor(%d) has max SST value of %f for fluor(%d)\n",idx,mxv,mxi);
  SST_[1][1]=1.0f; SST_[1][2]=0.4f; SST_[1][3]=0.0f; SST_[1][4]=0.3f;
  SST_[2][1]=0.5f; SST_[2][2]=1.0f; SST_[2][3]=0.3f; SST_[2][4]=0.4f;
  SST_[3][1]=0.5f; SST_[3][2]=0.3f; SST_[3][3]=1.0f; SST_[3][4]=1.0f;
  SST_[4][1]=0.8f; SST_[4][2]=0.3f; SST_[4][3]=0.1f; SST_[4][4]=0.8f;
  ::gaussj( SST_, nfluor_, CNT, 1 );
  break;
  }
 }
#endif
```

```
    ::free_matrix( CNT,   1,nfluor_, 1,1);
    return 1;
  }

/*****************************************************************************
 * FILE:        RatioBin.hxx
 * AUTHOR:  Andy Marks
 * Copyright © 1996 University of Utah
 */
#if !defined(_RATIOBIN_HXX_)
#define _RATIOBIN_HXX_
static int    const MAXPTRN = 20;
static double const FLTRK   = 12.0;
static double const MAXEUCD = 0.50;
class RatioPattern
{
public:
 RatioPattern( int nfluor=4 );
 ~RatioPattern() {};
 int add(double const* obs, int bandNr);
 double chm(int rdx, int cdx);
 void debug() const;
private:
 int nfluor_;
 int npatrn_;
 int chmrdy_;
 struct PATRN {
  int   nentry;
  double ratio[4];
  double rnsum[4];
  double r34;
```

```
    short  t3,
           t4,
           frstAtBand,
           lastAtBand;
  } patrn_[MAXPTRN];
  double chm_[4][4];
  void sortOnNentry_();
  void sortOnR34_();
  int orderIt_();
};
class RatioBin
{
public:
  RatioBin( int nsmpl=4 );
  RatioBin( RatioBin const& rhs );
  RatioBin const& operator=( RatioBin const& rhs);
  ~RatioBin();
  int classify( double const* smpls, int scanline );
  int analyze();
  double sst(short row, short col) const { return SST_[row][col]; }
  void debug(unsigned lvl=0) const;
private:
  void release_();
  int          nfluor_;
  float** CHM_;
  float** SST_;
  RatioPattern rp_;
};
#endif

/************************************************  ** ********** **
```

```
 * FILE:        RdrOut.cxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
#include <nrc/Complex.hxx>
#include <basecall/sw.hxx>
SSNODE::SSNODE() :
 fltwid_(0), start_(0), finish_(0), SWold_(0), thresh_(0.0f)
{
}
void
SSNODE::debug() const
{
  ::printf("SSNODE @ %p\n",(void*)this);
  ::printf("  fltwid  = %4d\n",fltwid());
  ::printf("  SWold   = %4d\n",SWold());
  ::printf("  start   = %4d\n",start());
  ::printf("  finish  = %4d\n",finish());
  ::printf("  rdlen   = %4d\n",rdlen());
  ::printf("  thresh  = %4.2f\n",thresh());
}
QualCtrl::QualCtrl() :
 tbgn_(0), tend_(0), nseg_(0), skipseq_(NULL)
{
  for(int idx=0;idx<MAXSEG;idx++)
   bspac_[idx] = fbwv_[idx] = 0;
}
char const*
QualCtrl::shft( int idx0, char* buf, int buflen ) const
{
```

```
    if((buflen>=15) && (idx0<nseg())) {
      ShftVect s = shft(idx0);
      ::sprintf( buf, "[%2hd,%2hd,%2hd,%2hd]", s.s(1),s.s(2),s.s(3),s.s(4));
      }
    else
      ::strcpy( buf, "?" );
    return (char const*)buf;
  }
  QualCtrl::QualCtrl(QualCtrl const& rhs) :
   tbgn_(0), tend_(0), nseg_(0), skipseq_(NULL)
  {
    *this = rhs;
  }
  QualCtrl const&
  QualCtrl::operator=(QualCtrl const& rhs)
  {
    if(this != &rhs) {
     for(int idx=0;idx<MAXSEG;idx++) {
      shft_[idx]  = rhs.shft_[idx];
      fbwv_[idx]  = rhs.fbwv_[idx];
      bspac_[idx] = rhs.bspac_[idx];
      }
     tbgn_   = rhs.tbgn_;
     tend_   = rhs.tend_;
     nseg_   = rhs.nseg_;
     cutdata_ = rhs.cutdata_;
     if(skipseq_) delete [] skipseq_;
     skipseq_ = new char[ 1+::strlen(rhs.skipseq_) ];
     if(skipseq_) ::strcpy( skipseq_, rhs.skipseq_ );
     }
    return *this;
```

```
}
QualCtrl::~QualCtrl()
{
  if(skipseq_) {
    delete [] skipseq_;
    skipseq_ = NULL;
    }
}
void
QualCtrl::debug() const
{
  ::printf("QualCtrl @ %p\n",(void const*)this);
  ::printf("  elapsed=%d(Sec)\n", tend_-tbgn_);
  cutdata_.debug();
  for(int idx=0;idx<nseg_;idx++) {
    ::printf("  idx=%d: fbw=%2d, bspac=%2d\t",idx,fbwv_[idx],bspac_[idx]);
    shft_[idx].debug();
    }
}
RdrOut::RdrOut( Wvfm const& in ) :
 overlayIdx_(1), oiS1_( in.bgni()+1 ), oPosLen_(0), oPos_(NULL), previ_(1)
{
  ::strcpy( lnordr_, in.lnordr() );
  for(int idx=1;idx<=OVRLAP;idx++)
    FM_[idx] = (-1.0/double(OVRLAP-1))*idx+(double(OVRLAP)/double(OVRLAP-1));
}
RdrOut::~RdrOut()
{
  if(NULL != oPos_) {
    ::free_ivector( oPos_, 1,oPosLen_);
    oPos_ = NULL;
```

```
    oPosLen_ = 0;
    }
  }
  int
  RdrOut::add( int passnr, int fbw, int medSP, SegRead const& sr )
  {
    int rv;
    qualctrl_.shft( passnr-1, sr.nsv() );
    qualctrl_.fbw( passnr-1, fbw );
    qualctrl_.bspac( passnr-1, medSP );
    qualctrl_.nseg( passnr );
    if(1 == passnr) {
     bandStats_ = sr.bandStats();
     traceOut_ = *sr.wvfm();
     rv = 1;
     }
    else if(1 == (rv = was_at( sr, passnr )))
     join( sr );
    if(rv) {
     if(NULL != oPos_)
      ::free_ivector( oPos_, 1,oPosLen_ );
     oPosLen_ = sr.bandStats().len();
     if((0==oPosLen_) || (NULL == (oPos_=::ivector(1,oPosLen_))))
      rv = 0;
     else for(int idx=1;idx<=oPosLen_;idx++)
      oPos_[idx] = sr.bandStats().posn(idx-1);
     }
    return rv;
  }
  void
  RdrOut::join( SegRead const& sr )
```

```
{
  int lhsEnd0, rhsBgn0,oscnl,nscnl, lShortFall,rShortFall;
  lhsEnd0 = bandStats_.posn(overlayIdx_-1) + 1 - OVRLAP/2;
  rhsBgn0 = sr.bandStats().posn(previ_-1) + 1 - OVRLAP/2;
  lShortFall = OVRLAP-(wvfm().rows()-lhsEnd0+1);
  rShortFall = 1-rhsBgn0;
  if(lShortFall > 0) {
   lhsEnd0 -= lShortFall;
   rhsBgn0 -= lShortFall;
   }
  else if(rShortFall > 0) {
   lhsEnd0 += rShortFall;
   rhsBgn0 += rShortFall;
   }
  for(int idx=1; idx<=OVRLAP; idx++) {
   double osf = FM_[idx], nsf = FM_[OVRLAP-idx+1];
   oscnl = lhsEnd0+idx-1;
   nscnl = rhsBgn0+idx-1;
   for(int lane=1;lane<=4;lane++) {
    double ov, nv;
    ov = wvfm().sc_la( oscnl, lane );
    nv = sr.wvfm()->sc_la( nscnl, lane );
    wvfm().sc_la_set( oscnl,lane, osf*ov + nsf*nv );
    }
   }
  wvfm().append( *sr.wvfm(), oscnl, nscnl+1 );
  bandstat().append( sr.bandStats(),overlayIdx_-1,previ_ );
}
void
RdrOut::closesTo(int wasAt, int& dist, int& j) const
{
```

```
        int lodx=1, hidx=oPosLen_, tsti, tstp, htsti, ltsti;
        ltsti = lodx;
        htsti = hidx;
        while(lodx <= hidx) {
          tstp = oPos_[ tsti = (hidx+lodx)/2 ];
          if(wasAt == tstp) {
            dist = 0;
            j = tsti;
            return;
            }
          else if(wasAt < tstp) {
            htsti = tsti--;
            hidx = tsti;
            }
          else {
            ltsti = tsti++;
            lodx = tsti;
            }
          }
        int dlo = abs( wasAt - oPos_[ltsti] ),
            dhi = abs( oPos_[htsti] - wasAt );
        if(dlo < dhi) { dist = dlo; j = ltsti; }
        else          { dist = dhi; j = htsti; }
      }
      int
      RdrOut::was_at( SegRead const& sr, int n )
      {
        static int const CHKORDR[] = {0,5,6,4,7,3,8,2,9,1,10};
      #  define NCHK ((sizeof(CHKORDR)/sizeof(CHKORDR[0]))-1)
        int mindist=4, xlation[3], cdx, K;
        int Nband = sr.bandStats().len():
```

```
    int rv = 0;
    K = sr.iS1() - (oiS1_ + (n-2)*1900);
    for(cdx=1; cdx<=NCHK; cdx++) {
     int Cdx = CHKORDR[ cdx ];
     for(int lane=0;lane<4;lane++) {
      if((Cdx<=Nband)&& (lnordr_[lane] == sr.bandStats().call(Cdx-1)))
        break;
      }
     if(5 != ++lane) {
      int wasAt, dist,j;
      wasAt = (sr.bandStats().posn(Cdx-1) - qualctrl().shft(n-1).s(lane)) +
           (K + qualctrl().shft(n-2).s(lane));
      closesTo( wasAt, dist,j );
      if(dist < mindist) {
       rv = 1;
       xlation[1] = Cdx;
       xlation[2] = j;
       if(0 == (mindist=dist))
        break;
       }
      }
     }
    if(1 == rv) {
     overlayIdx_ += (xlation[2]-previ_);
     previ_ = xlation[1];
     }
    return rv;
}
void
RdrOut::Edit( char const* preamble )
{
```

```
    int lcutscnl, rcutscnl, NB;
    NB = bandStats_.len();
    lcutscnl = bandStats_.posn(0);
    rcutscnl = bandStats_.posn(NB-1);
    bandqual();
    pickcuts( preamble );
    qualctrl_.stopTimer();
}
static int const FW[] = { 7, 9, 11, 13, 15 };
static float const TH[] =
{ 0.74f, 0.75f, 0.76f, 0.77f, 0.78f, 0.79f, 0.80f };
static int const NW = (sizeof(FW)/sizeof(FW[0])),
            NT = (sizeof(TH)/sizeof(TH[0]));
void
RdrOut::cutoff(int fdx, SSNODE& ss, int FFTSZ) const
{
  int NB = bandStats_.len();
  int runlen=0, mxrl=0;
  ss.fltwid( FW[ fdx ] );
  if(ss.fltwid() <= FFTSZ) {
   int LHS = (1+FW[fdx])/2, RHS = FW[fdx]-LHS;
   int anyGood=0, anyL2H=0, anyH2L=0;
   int idx, jdx, good, goodM1=0;
   for(idx=1;idx<=2*LHS;idx+=2) {
    HF_[ idx ] = 1.0/double(ss.fltwid());
    HF_[idx+1] = 0.0;
    }
   for(;idx<=2*(FFTSZ-RHS);idx++)
    HF_[ idx ] = 0.0;
   for(;idx<=2*FFTSZ;idx+=2) {
      HF_[ idx ] = 1.0/double(ss.fltwid());
```

```
 HF_[idx+1] = 0.0;
 }
::dfour1( HF_, FFTSZ, 1);
::dCMul( XF_, HF_, AF_, FFTSZ );
::dCMul( TF_, HF_, BF_, FFTSZ );
::dfour1( AF_, FFTSZ, -1);
::dfour1( BF_, FFTSZ, -1);
for(jdx=idx=1;jdx<=2*NB;idx++,jdx+=2) {
 if(good = (AF_[jdx] >= BF_[jdx]))
  anyGood++;
 if(1 != idx) {
  int dg=good-goodM1;
  if(dg>0) nL2H_[++anyL2H] = idx-1;
  else if(dg<0)    nH2L_[++anyH2L] = idx-1;
  }
 goodM1 = good;
 }
if(anyGood) {
 if(anyGood == NB) {
  ss.start( 1 );
  ss.finish( NB );
  }
 else if(0==anyL2H) {
  ss.start( 1 );
  ss.finish( nH2L_[1] );
  }
 else if(0==anyH2L) {
  ss.start( 1+nL2H_[1] );
  ss.finish( NB );
  }
 else {
```

```
        int *mL2H, *mH2L, mLn=anyL2H, mHn=anyH2L, lastH2L, lastL2H;
        mL2H = ::ivector(1,anyL2H+1);
        mH2L = ::ivector(1,anyH2L+1);
        lastH2L = nH2L_[ mHn ];
        lastL2H = nL2H_[ mLn ];
        if(nH2L_[1] < nL2H_[1]) {
         mL2H[1] = 1;
         for(idx=1;idx<=mLn;idx++)      mL2H[idx+1] = nL2H_[idx];
         mLn++;
         }
        else for(idx=1;idx<=anyL2H;idx++) mL2H[idx]   = nL2H_[idx];
        for(idx=1;idx<=mHn;idx++) mH2L[idx] = nH2L_[idx];
        if(lastH2L < lastL2H)
         mH2L[++mHn] = NB;
        if(mHn == mLn) {
         for(idx=1;idx<=mHn;idx++) {
          runlen = mH2L[idx]-mL2H[idx];
          if(runlen > mxrl) {
           mxrl = runlen;
           ss.start( mL2H[idx] );
           ss.finish( mH2L[idx] );
           }
          }
        ::free_ivector(mL2H, 1,anyL2H+1 );
        ::free_ivector(mH2L, 1,anyH2L+1 );
        }
       }
      }
     }
}
void
```

```
RdrOut::pickcuts( char const* preamble )
{
  static float const WGHT[6][9] =
  {
{0.8944f,0.9076f,0.9208f,0.9340f,0.9472f,0.9604f,0.9736f,0.9868f,1.0000f},
{0.8755f,0.8885f,0.9014f,0.9143f,0.9272f,0.9401f,0.9530f,0.9660f,0.9789f},
{0.8567f,0.8693f,0.8819f,0.8946f,0.9072f,0.9199f,0.9325f,0.9451f,0.9578f},
{0.8378f,0.8501f,0.8625f,0.8749f,0.8872f,0.8996f,0.9119f,0.9243f,0.9367f},
{0.8189f,0.8310f,0.8430f,0.8551f,0.8672f,0.8793f,0.8914f,0.9035f,0.9155f},
{0.8000f,0.8118f,0.8236f,0.8354f,0.8472f,0.8590f,0.8708f,0.8826f,0.8944f}
  };
  int jdx, idx, FFTSZ, NB = bandStats_.len(), mxrdlen=0;
  FFTSZ = int( ::pow( 2.0, ceil(::log(double(NB))/::log(2.0)) ) );
  XF_ = ::dvector( 1, FFTSZ*2 );
  TF_ = ::dvector( 1, FFTSZ*2 );
  HF_ = ::dvector( 1, FFTSZ*2 );
  AF_ = ::dvector( 1, FFTSZ*2 );
  BF_ = ::dvector( 1, FFTSZ*2 );
  nH2L_ = ::ivector( 1,NB );
  nL2H_ = ::ivector( 1,NB );
  for(jdx=idx=1;idx<2*NB;jdx++,idx+=2) {
   XF_[ idx ] = double( bandStats_.qual(jdx-1) );
   XF_[ idx+1 ] = 0.0;
   }
  for(; idx<=2*FFTSZ; idx++)
   XF_[ idx ] = 0.0;
  ::dfour1( XF_, FFTSZ, 1);
    for(int tdx=0;tdx<NT;tdx++) {
   for(idx=1;idx<2*NB;idx+=2) {
    TF_[ idx ] = double( TH[tdx] );
    TF_[ idx+1 ] = 0.0;
```

```
    }
   for(; idx<=2*FFTSZ; idx++) TF_[ idx ] = 0.0;
   ::dfour1( TF_, FFTSZ, 1);
   for(int fdx=0;fdx<NW;fdx++) {
    SSNODE ss;
    ss.thresh( TH[tdx] );
    cutoff( fdx, ss, FFTSZ );
    int wt_rdlen = int(WGHT[fdx][tdx] * ss.rdlen());
    if(mxrdlen < wt_rdlen) {
     mxrdlen = wt_rdlen;
     qualctrl_.cutdata( ss );
     }
    }
   }
  ::free_dvector( XF_, 1,2*FFTSZ );
  ::free_dvector( HF_, 1,2*FFTSZ );
  ::free_dvector( TF_, 1,2*FFTSZ );
  ::free_dvector( AF_, 1,2*FFTSZ );
  ::free_dvector( BF_, 1,2*FFTSZ );
  ::free_ivector( nH2L_, 1,NB );
  ::free_ivector( nL2H_, 1,NB );
  if(NULL != preamble) {
   int lpre2x = ::strlen(preamble) * 2;
   int lobs = bandstat().len();
   int len = (lpre2x<lobs)? lpre2x: lobs;
   if(qualctrl().cutdata().start() < len) {
    char* obscopy = new char[ len+1 ];
    if(NULL != obscopy) {
     static int const MAGIC_NR = 4;
     int idx, s;
     for(idx=0;idx<len;idx++)
```

```
            obscopy[idx] = bandstat().call(idx);
          obscopy[idx] = '\0';
          SW swalign( preamble, obscopy );
          s = qualctrl().cutdata().start();
          if(swalign.hcoord() > s) {
           int hslen = ::strlen(swalign.hout());
           if(hslen >= MAGIC_NR) {
            int hitval = (swalign.score()*swalign.score())/hslen;
            if(hitval > MAGIC_NR) {
             qualctrl().cutdata().SWold( s );
             qualctrl().cutdata().start( swalign.hcoord()+1 );
             }
            }
           }
          delete [] obscopy;
          }
         }
        }
      }
      void
      RdrOut::Beautify(int fbool,int bbool)
      {
        if(1==fbool) flatten_();
        if(1==bbool) minNegSwing_();
      }
      void
      RdrOut::flatten_()
      {
        int idx, jdx, kdx, N, FFTSZ;
        double *sn, *hn;
        N = traceOut_.endi()-traceOut_.bgni()+1;
```

```
    FFTSZ = int(::pow(2.0,::ceil(::log(double(N+129-1))/::log(2.0))));
    sn = ::dvector(1,2*FFTSZ);
    hn = ::dvector(1,2*FFTSZ);
    for(idx=1; idx<=2*FFTSZ; idx++)
      hn[idx] = sn[idx] = 0.0;
    for(jdx=1,idx=traceOut_.bgni(); jdx<=2*N; idx++, jdx+=2)
      sn[jdx] = traceOut_.envv(idx);
    for(kdx=2*FFTSZ-1,jdx=idx=1; idx<=128; idx++,jdx+=2,kdx-=2)
      hn[jdx] = hn[kdx] = 1.0/129.0;
    hn[jdx] = 1.0/129.0;
    ::dfour1( sn, FFTSZ, 1 );
    ::dfour1( hn, FFTSZ, 1 );
    ::dCMul( sn, hn, sn, FFTSZ );
    ::dfour1( sn, FFTSZ, -1 );
    for(kdx=1,idx=traceOut_.bgni(); idx<=traceOut_.endi(); idx++,kdx+=2) {
      double sf = double(FFTSZ)/(DBL_EPSILON+sn[kdx]);
      for(jdx=1;jdx<=4;jdx++)
        traceOut_.sc_la_mul(idx,jdx,sf);
      }
    ::free_dvector(sn,1,2*FFTSZ);
    ::free_dvector(hn,1,2*FFTSZ);
  }
  void
  RdrOut::minNegSwing_()
  {
    for(int lnr=1;lnr<=traceOut_.lanes();lnr++) {
      double minv = 0.0;
      int bgn=traceOut_.bgni(), end=traceOut_.endi();
      for(int snr=bgn; snr<=end; snr++) {
        double v = traceOut_.sc_la(snr,lnr);
        if(v < minv) minv = v;
```

```
        }
       if(0.0 > minv) {
        double sf = -0.02/minv;
        for(snr=bgn;snr<=end;snr++)
         if(traceOut_.sc_la(snr,lnr) < 0.0)
           traceOut_.sc_la_mul(snr,lnr,sf);
       }
      }
    }
    int
    RdrOut::avgqual() const
    {
      float aq = 0.0f;
      int bgn = qualctrl().cutdata().start(),
        end = qualctrl().cutdata().finish();
      BandStatArray const& bs = bandstat();
      for(int idx=bgn;idx<end;idx++)
       aq += int(100.0f * bs.qual(idx));
      return int(0.5f + aq/float(end-bgn+1));
    }
    int
    RdrOut::percentN() const
    {
      BandStatArray const& bs = bandstat();
      int bgn = qualctrl().cutdata().start(),
        end = qualctrl().cutdata().finish(),
        ambig  = 0;
      for(int idx=bgn;idx<end;idx++)
       if(lnordr_[4] == bs.call(idx))
        ambig++;
      return int(0.5f + 100.0f*float(ambig)/float(end-bgn+1));
```

```
}
char const*
RdrOut::sequence( char* buf, int buflen ) const
{
    BandStatArray const& bs = bandstat();
    int len = (bs.len()<buflen)? bs.len(): buflen;
    for(int idx=0;idx<len;idx++)
      buf[idx] = bs.call(idx);
    buf[idx] = '\0';
    return (char const*)buf;
}
void
RdrOut::debug() const
{
    ::printf("RdrOut at %p\n",(void const*)this);
    ::printf("  oiS1 = %d\n",iS1());
    ::printf("  average quality = %2d,  %%ambig = %2d\n",avgqual(),percentN());
    qualctrl_.debug();
    bandStats_.debug();
    traceOut_.debug();
}

/*************************************************************************
 * FILE:        RdrOut.hxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#ifndef _RDROUT_HXX_
#define _RDROUT_HXX_
#include <time.h>
#include <basecall/Metrics.hxx>
```

```
static int const MAXSEG  = 6;
static int const OVRLAP = 20;
#if defined(WIN32)
class _declspec( dllexport ) SSNODE
#else
class SSNODE
#endif
{
public:
  SSNODE();
  ~SSNODE() {;}
  int fltwid()   const { return fltwid_; }
  int start()    const { return start_; }
  int finish()   const { return finish_; }
  int rdlen()    const { return finish_-start_+1; }
  int SWold()    const { return SWold_; }
  float thresh() const { return thresh_; }
  void fltwid(int w)   { fltwid_ = w; }
  void start(int s)    { start_  = s; }
  void finish(int f)   { finish_ = f; }
  void SWold(int s)    { SWold_  = s; }
  void thresh(float t) { thresh_ = t; }
  void debug() const;
private:
  int fltwid_,
    start_,
    finish_,
    SWold_;
  float thresh_;
};
#if defined(WIN32)
```

```
class _declspec( dllexport) QualCtrl
#else
class QualCtrl
#endif
{
public:
  QualCtrl();
  QualCtrl( QualCtrl const& rhs );
  QualCtrl const& operator=( QualCtrl const& rhs );
  ~QualCtrl();
  void shft(  int idx0, ShftVect const& sv )
   { if(idx0<MAXSEG) shft_[idx0] = sv; }
  void fbw( int idx0, int val )
   { if(idx0<MAXSEG) fbwv_[idx0] = val; }
  void bspac( int idx0, int val )
   { if(idx0<MAXSEG) bspac_[idx0] = val; }
  void nseg( int v ) { nseg_ = v; }
  void cutdata( SSNODE const& v )  { cutdata_ = v; }
  void ignoredSeq( char const* seq );
  void startTimer()          { tbgn_ = time((time_t*)NULL); }
  void stopTimer()           { tend_ = time((time_t*)NULL); }
  int nseg()           const { return nseg_; }
  SSNODE&     cutdata()      { return cutdata_; }
  SSNODE const& cutdata()    const { return cutdata_; }
  char const*  ignoredSeq()  const { return NULL; }
  char const* shft( int idx0, char* buf, int buflen ) const;
  ShftVect shft( int idx0 )     const
   { return (idx0<nseg())? shft_[idx0]: shft_[0]; }
  int     fbw(  int idx0 )      const
   { return (idx0<MAXSEG)? fbwv_[idx0]: -1; }
  int     bspac( int idx0 )     const
```

```
    { return (idx0<MAXSEG)? bspac_[idx0]: -1; }
    time_t runtime()          const { return tend_-tbgn_+1; }
    void debug() const;
  private:
    ShftVect shft_[MAXSEG];
    int      fbwv_[MAXSEG];
    int      bspac_[MAXSEG];
    time_t   tbgn_,
             tend_;
    int      nseg_;
    SSNODE   cutdata_;
    char*    skipseq_;
  };
  #if defined(WIN32)
  class _declspec( dllexport) RdrOut
  #else
  class RdrOut
  #endif
  {
  public:
    RdrOut( Wvfm const& in );
    ~RdrOut();
    void iS1( int iS1 ) { oiS1_ = iS1; }
    int add( int passnr, int fbw, int medSP, SegRead const& sr );
    int      iS1()          const { return oiS1_; }
    int      avgqual()      const;
    int      percentN()     const;
    BandStatArray&      bandstat()      { return bandStats_; }
    BandStatArray const& bandstat() const { return bandStats_; }
    char const* sequence(char* buffer,int buflen) const;
    Wvfm&            wvfm()         { return traceOut_; }
```

```
    Wvfm const&       wvfm()    const { return traceOut_; }
    QualCtrl&         qualctrl()     { return qualctrl_; }
    QualCtrl const&   qualctrl() const { return qualctrl_; }
    void Edit( char const* preamble = NULL );
    void Beautify( int fbool, int bbool);
    void debug() const;
private:
    int       overlayIdx_,
              oiS1_,
              oPosLen_,
              *oPos_,
              previ_;
    char      lnordr_[6];
    BandStatArray bandStats_;
    Wvfm      traceOut_;
    QualCtrl  qualctrl_;
    double    FM_[1+OVRLAP];
    double    *XF_,
              *TF_,
              *HF_,
              *AF_,
              *BF_;
    int       *nH2L_,
              *nL2H_;
    int was_at(SegRead const& sr, int passnr);
    void closesTo(int wasAt, int& dist, int& j) const;
    void join( SegRead const& sr );
    void bandqual();
    void pickcuts( char const* preamble );
    void cutoff( int fdx, SSNODE& ss, int FFTSZ ) const;
    void flatten_();
```

```
    void minNegSwing_();
};
#endif

/*****************************************************************************
 * FILE:        SegRead.cxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
SegRead::SegRead( int iS1, int fluor ) :
 status_(STS_INITD), iS1_( iS1 ), nWvf_(NULL), pmb_(NULL)
{
   pmb_ = new MB(fluor);
}
SegRead::~SegRead()
{
   if(nWvf_)   { delete nWvf_; nWvf_ = NULL; }
   if(pmb_)    { delete pmb_;  pmb_ = NULL; }
}
SegRead::SegRead( SegRead const& rhs ) :
 status_(STS_UNINITD), iS1_(0), nWvf_(NULL), pmb_(NULL)
{
   *this = rhs;
}
SegRead const&
SegRead::operator=( SegRead const& rhs )
{
   if(this != &rhs) {
      if(nWvf_)  { delete nWvf_; nWvf_ = NULL; }
      if(pmb_)   { delete pmb_; pmb_ = NULL; }
```

```
        nWvf_  = new Wvfm( *rhs.nWvf_ );
        pmb_   = new MB( *rhs.pmb_ );
        iS1_   = rhs.iS1_;
        nsv_   = rhs.nsv_;
        bsa_   = rhs.bsa_;
        status_ = rhs.status_;
        }
      return *this;
    }
    void
    SegRead::wvfm( Wvfm const& rhs )
    {
      if(NULL != nWvf_) { delete nWvf_; nWvf_ = NULL; }
      nWvf_ = new Wvfm( rhs );
    }
    float*
    SegRead::xbndara_( PKDET const& pks ) const
    {
      int idx, N = pks.npk();
      float* xbnd;
      if(NULL != (xbnd = ::vector(1,N)))
       for(idx=1; idx<=N; idx++)
        xbnd[ idx ] = float(nWvf_->xbnd( pks.bmid( idx ) ));
      return xbnd;
    }
    float*
    SegRead::buzzara_( PKDET const& pks ) const
    {
      int idx, N = pks.npk();
      float* buzz;
      if(NULL != (buzz = ::vector(1,N)))
```

```
    for(idx=1; idx<=N; idx++)
      buzz[ idx ] = float(nWvf_->buzz( pks.bmid( idx ) ));
    return buzz;
  }
int
SegRead::peakdet( int npts, PKDET& putative ) const
{
  double vm1,v;
  int *ppk=NULL,Np=0, *ptr=NULL,Nt=0, idx;
  Wvfm const& wvfm = *nWvf_;
  short MAXSV = nsv_.maxshft();
  vm1 = wvfm.envv( MAXSV+1 );
   v = wvfm.envv( MAXSV+2 );
  ppk = ::ivector( 1, npts );
  ptr = ::ivector( 1, npts );
  if(NULL == ppk || NULL == ptr)
   return 0;
  enum STATE { ST_UK, ST_UP, ST_DN } st = ST_UK;
  for(idx = MAXSV+2; idx <= npts; idx++ ) {
   switch(st) {
    case ST_UK:
     if(v > vm1)     st=ST_UP;
     else if(v < vm1) st=ST_DN;
     break;
    case ST_UP:
     if(v < vm1) { st = ST_DN; ppk[ ++Np ] = idx-1; }
     break;
    case ST_DN:
     if(v > vm1) { st = ST_UP; ptr[ ++Nt ] = idx-1; }
     break;
    }
```

```
        vm1 = v;
        v = wvfm.envv( idx+1 );
        }
      putative.set( ppk, Np, ptr, Nt );
      ::free_ivector( ppk, 1,npts );
      ::free_ivector( ptr, 1,npts );
      return 1;
    }
    void
    SegRead::maxlanecode_( int have, int* bcodes ) const
    {
      short MAXSV = nsv().maxshft();
      for(int sdx=MAXSV+1; sdx<=int(have); sdx++) {
       double THR = 0.80*nWvf_->envv(sdx);
       int ldx, code = 0;
       for(ldx=1; 5!=code && ldx<=int(nWvf_->lanes()); ldx++)
        if(nWvf_->sc_la(sdx,ldx) >= THR)
          code = (0==code)? ldx: 5;
       bcodes[ sdx ] = code;
       }
    }
    static void
    dfliplr(double* p, int n)
    {
      int idx, ndx;
      double t;
      for(idx=1,ndx=n;idx<=(n/2);idx++,ndx--)
       { t = p[idx]; p[idx] = p[ndx]; p[ndx] = t; }
    }
    int
    SegRead::setBandStats(PKDET const& final,int FBW,char const* seq,int pass2)
```

```
{
 int NPK = final.npk(), *insSP=NULL, *insWD=NULL;
 int rv = 1;
 double f_sigma = (double(FBW-1)*2.0*NR_PI)/2048.0;
 double t_sigma = 1.0/f_sigma;
 int M = (final.medWid() + 1)/2;
 float PATTCOEF[4], *xbnd, *buzz;
 if(1 == pass2) {
  double *PX = ::dvector(1,M), *PY = ::dvector(1,M);
  int sts;
  if(!PX || !PY) return 0;
  for(int pdx=1;pdx<=M;pdx++) {
   double v;
   PX[pdx] = double(pdx);
   v = (PX[pdx]-((double(M)+1.0)/2.0))/(t_sigma/2.0);
   PY[pdx] = ::exp(-0.5*v*v);
   }
  sts = ::dquadratic( PX,PY, M, PATTCOEF );
  ::free_dvector(PX,1,M);
  ::free_dvector(PY,1,M);
  if(1 != sts)  return 0;
  }
 if(NULL == (insSP = ::ivector( 1, NPK ))) {
  status_ = STS_NO_MEM;
  return 0;
  }
 if(1 != ::insMetric( final.bmid(), final.lgap(), insSP, NPK )) {
  status_ = STS_TOO_FEW;
  ::free_ivector(insSP,1,NPK);
  return 0;
  }
```

```
if(NULL == (insWD = ::ivector( 1, NPK ))) {
 status_ = STS_NO_MEM;
 ::free_ivector(insSP,1,NPK);
 return 0;
 }
if(1 != ::insMetric( final.bmid(), final.bwid(), insWD, NPK )) {
 status_ = STS_TOO_FEW;
 ::free_ivector(insSP,1,NPK);
 ::free_ivector(insWD,1,NPK);
 return 0;
 }
if(NULL == (xbnd = xbndara_( final ))) {
 status_ = STS_NO_MEM;
 ::free_ivector(insSP,1,NPK);
 ::free_ivector(insWD,1,NPK);
 return 0;
 }
if(NULL == (buzz = buzzara_( final ))) {
 status_ = STS_NO_MEM;
 ::free_ivector(insSP,1,NPK);
 ::free_ivector(insWD,1,NPK);
 ::free_vector(xbnd,1,NPK);
 return 0;
 }
BandStatArray& bsa = bandStats();
bsa.init( NPK );
for(int idx0 = 0; idx0 < NPK; idx0++) {
 int idx1  = idx0+1, scanl = final.bmid( idx1 );
 float onsv, ofsv;
 bsa.ntnr( idx0, idx1 );
 bsa.posn( idx0, scanl );
```

```
bsa.hght( idx0, float(nWvf_->envv( scanl ) ));
onsv = float(nWvf_->envv(final.bbgn(idx1)));
ofsv = float(nWvf_->envv(final.bend(idx1)));
bsa.lowv( idx0, onsv<ofsv?onsv:ofsv );
bsa.xbnd( idx0, xbnd[ idx1 ] );
bsa.buzz( idx0, buzz[ idx1 ] );
bsa.insr( idx0, final.ins(idx1) );
bsa.call( idx0, seq[ idx1 ] );
if(pass2) {
 double cc = -1.0;
 int N = final.bwid(idx1);
 if(N>4) {
  double *BX, *BY, miny = 1000.0, cclr, ccrl;
  float bandcoef[4];
  int wdx, scanl = final.bbgn(idx1);
  BX = ::dvector(1,N);
  BY = ::dvector(1,N);
  for(wdx=1;wdx<=N;wdx++) {
   BX[wdx] = double(wdx);
   BY[wdx] = nWvf_->envv( scanl++ );
   if(BY[wdx] < miny) miny = BY[wdx];
   }
  for(wdx=1;wdx<=N;wdx++)
   BY[wdx] -= miny;
  if(1 != ::dquadratic( BX, BY, N, bandcoef )) {
   rv = 0;
   status_ = STS_TOO_FEW;
   goto bugout;
   }
  cclr = ::corrcoef( PATTCOEF, bandcoef, 4 );
  ::dfliplr( BY, N );
```

```
if(1 != ::dquadratic( BX, BY, N, bandcoef )) {
 rv = 0;
 status_ = STS_TOO_FEW;
 goto bugout;
 }
ccrl = ::corrcoef( PATTCOEF, bandcoef, 4 );
cc = (cclr>ccrl)? cclr: ccrl;
if(isnan(cc)) cc = -1.0;
::free_dvector(BX,1,N);
::free_dvector(BY,1,N);
 }
bsa.shap( idx0, float(cc) );
double width, iwidth;
width  = double( final.bwid(idx1) );
iwidth = double( insWD[ idx1 ] );
bsa.widt( idx0,  float(width/iwidth - 1.0) );
bsa.bbgn( idx0, final.bbgn(idx1) );
bsa.bend( idx0, final.bend(idx1) );
float igap,slope,intercept, lfgap,rtgap,biggap,smlgap;
igap = float(insSP[idx1]);
if(0.0f == igap) {
 bsa.lgap( idx0, 0.5f );
 bsa.sgap( idx0, 0.5f );
 rv = 0;
 goto bugout;
 }
else {
 igap  = float(insSP[idx1]);
 slope = igap/4.0f;
 intercept = igap/2.0f;
 lfgap = float( final.lgap(idx1) );
```

```
        rtgap = float( final.rgap(idx1) );
        if(lfgap>rtgap) { biggap = lfgap; smlgap = rtgap; }
        else        { biggap = rtgap; smlgap = lfgap; }
        if(biggap < slope) biggap = intercept - biggap*slope;
        if(smlgap < slope) smlgap = intercept - smlgap*slope;
#if !defined(_WIN32)
        bsa.lgap( idx0, fmod(biggap,igap)/igap );
        bsa.sgap( idx0, fmod(smlgap,igap)/igap );
#else
        bsa.lgap( idx0, float(fmod(biggap,igap)/igap) );
        bsa.sgap( idx0, float(fmod(smlgap,igap)/igap) );
#endif
      }
     }
    }
bugout:
  ::free_vector( buzz, 1,NPK ); buzz = NULL;
  ::free_vector( xbnd, 1,NPK ); xbnd = NULL;
  ::free_ivector( insWD, 1,NPK ); insWD = NULL;
  ::free_ivector( insSP, 1,NPK ); insSP = NULL;
  return rv;
}
void
SegRead::fbwlut( int spacing, int& fbw ) const
{
  if(spacing < SMLGAP)    spacing = SMLGAP;
  else if(spacing > BIGGAP) spacing = BIGGAP;
  fbw = pmb_->fbwlut_[ spacing ];
}

/************************************** ******************************
```

```
* FILE:      SegRead.hxx
* AUTHOR:    Andy Marks
* NOTE:      To use this easily include seqrdr.hxx, which
*            includes all the headers this needs.
* COPYRIGHT (c) 1996, University of Utah
*/
#ifndef _SEGREAD_HXX_
#define _SEGREAD_HXX_
static int const STATIC_BUF_SZ = NPTS/2;
class MB;
class PKDET;
class SegRead
{
public:
  enum Status { STS_UNINITD, STS_INITD, STS_NO_MEM, STS_BUF2SMALL,
                STS_TOO_FEW };
  SegRead( int iS1, int fluor );
  ~SegRead();
  SegRead( SegRead const& rhs );
  SegRead const& operator=( SegRead const& rhs );
  void iS1( int v )      { iS1_ = v; }
  void wvfm( Wvfm const& rhs );
  int              iS1()      const { return iS1_; }
  Wvfm*            wvfm()           { return nWvf_; }
  Wvfm const*      wvfm()     const { return nWvf_; }
  ShftVect&        nsv()            { return nsv_; }
  ShftVect const&  nsv()      const { return nsv_; }
  BandStatArray&   bandStats()      { return bsa_; }
  BandStatArray const&  bandStats() const { return bsa_; }
  void fbwlut( int spacing, int& fbw ) const;
  int fBandSpace( int& medSP ) const;
```

```
    int nrefine( char const* lnordr, int FBW, int have, int p2 );
    void blindeconv( Wvfm& sr, int Fbw );
    int xtranorm( Wvfm& bd, Wvfm const& source, int nscanl, int pass2 );
    int setBandStats(PKDET const& final,int FBW,char const* seq,int pass2);
    int peakdet( int npts, PKDET& putative ) const;
    Status status() const { return status_; }
    void debug() const;
private:
    Status status_;
    int       iS1_;
    Wvfm      *nWvf_;
    ShftVect  nsv_;
    BandStatArray bsa_;
    Band nband_[ STATIC_BUF_SZ ];
    char seq_ [ STATIC_BUF_SZ ];
    void maxlanecode_( int n, int* bcode ) const;
    float* xbndara_( PKDET const& putative ) const;
    float* buzzara_( PKDET const& putative ) const;
    int centroid_(int bgn, int end) const;
    MB        *pmb_;
};
#endif

/*****************************************************************************
 * FILE:      seqrdr.hxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#ifndef _SEQRDR_HXX_
#define _SEQRDR_HXX_
static int const NPTS = 2048;
```

```
#include <stdio.h>
#include <math.h>
#include <limits.h>
#include <float.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <nrc/nr.hxx>
#include <basecall/Wvfm.hxx>
#include <basecall/Metrics.hxx>
#include <basecall/RdrOut.hxx>
#include <basecall/AboutBQ.hxx>
#if defined(sun)
# include <ieeefp.h>
#endif
int d_cmp( void const*, void const* );
double corrcoef( float const* v1, float const* v2, int n );
int insMetric( int const* px, int const* py, int* ytmp, int N );
#if defined(__cplusplus)
extern "C" {
#endif
int omitokn( int const* i, float const* h, float const* l,
        int const* lg, int const* rg,
        float const* x, int n, float** o);
int gapcheck( int const* ig,int const* iw,int const* g,int const* w,
        char const* seq, int n, float** o);
#if defined(__cplusplus)
}
#endif
#endif
```

```
/************************************************************ * ********
 * FILE:        ShftVect.cxx
 * AUTHOR:  Andy Marks
 * Copyright (c) 1995,1996, University of Utah
 */
#include <basecall/mb.hxx>
static const int
  NCUBES = 6,
  M     = (27*NCUBES) - (NCUBES-1);
class Criterion
{
public:
   Criterion() : val_(0.0), idx_(0) {;}
   ~Criterion() {;}
   Criterion const& operator=( Criterion const& rhs);
   void   debug( int lvl = 0 ) const;
   double val_;
   int    idx_;
};
class ShftVects
{
public:
   ShftVects( int len );
   ShftVects( ShftVect const& sv );
   ShftVects( ShftVects const& rhs );
   ShftVects const& operator=( ShftVects const& rhs );
   ~ShftVects();
   ShftVect align( Wvfm const& t );
   void debug( int lvl = 0 ) const;
private:
   void evaluate( Wvfm const& t );
```

```
    void best( int nelem );
    int terminate();
    int    len_;
    ShftVect* svm_;
    Criterion* crit_;
    int    mni_, mxi_;
    int    loops_;
};
ShftVect
mcalign( Wvfm const& trace, ShftVect const& sv )
{
    ShftVects svm( sv );
    return svm.align( trace );
}
class TriVects
{
public:
    TriVects();
    TriVect const& operator[](int idx) const { return tvm_[idx]; }
    ~TriVects();
    void debug() const;
private:
    int len_;
    TriVect* tvm_;
};
Criterion const&
Criterion::operator=( Criterion const& rhs)
{
    if(this != &rhs) {
        idx_ = rhs.idx_;
        val_ = rhs.val_;
```

```
    }
  return *this;
}
void
Criterion::debug( int lvl ) const
{
  ::printf("Criterion: val=%8.3f idx=%2d\n",val_,idx_);
}
static const TriVects tvm;
TriVect::TriVect()
{
  t[0] = t[1] = t[2] = 0;
}
TriVect::TriVect( TriVect const& rhs )
{
 t[0] = rhs.t[0];
 t[1] = rhs.t[1];
 t[2] = rhs.t[2];
}
TriVect
TriVect::operator+( TriVect const& rhs ) const
{
  TriVect s( rhs );
  s.t[0] += t[0];
  s.t[1] += t[1];
  s.t[2] += t[2];
  return s;
}
void
TriVect::debug( int lvl ) const
{
```

```
  ::printf("TriVect: [%hd,%hd,%hd]\n",t[0],t[1],t[2]);
}
TriVects::TriVects() : len_(M), tvm_(NULL)
{
  static int const radii[] = { 4, 8, 12, 16, 20, 24 };
  int idx = 1;
  tvm_ = new TriVect[ len_ ];
  if(NULL == tvm_) {
   ::fprintf(stderr,"TriVects::TriVects() out of memory.\n");
   exit(1);
   }
  tvm_[0].t[0] = tvm_[0].t[1] = tvm_[0].t[2] = 0;
  for(int s=0;s<NCUBES;s++) {
   int ko[3];
   ko[0] = -radii[s]; ko[1] = 0; ko[2] = radii[s];
   for(int i0=0; i0<3; i0++)
    for(int i1=0; i1<3; i1++)
     for(int i2=0; i2<3; i2++)
      if(1!=i0 || 1!=i1 || 1!=i2) {
       tvm_[idx].t[0] = ko[i0];
       tvm_[idx].t[1] = ko[i1];
       tvm_[idx].t[2] = ko[i2];
       idx++;
       }
  }
}
TriVects::~TriVects()
{
 if(tvm_) { delete [] tvm_; tvm_ = NULL; len_ = 0; }
}
void
```

```
TriVects::debug() const
{
  ::printf("TriVects @ %p\n",(void*)this);
  ::printf("\tlen_ = %ld\n",len_);
  for(int idx = 0; idx < len_; idx++)
    tvm_[idx].debug();
}
ShftVect::ShftVect()
{
  s_[0] = s_[1] = s_[2] = s_[3] = 0;
}
ShftVect::ShftVect(short v1, short v2, short v3, short v4)
{
  s_[0] = v1; s_[1] = v2; s_[2] = v3; s_[3] = v4;
}
ShftVect::ShftVect( TriVect const& tv )
{
  int min = 1000;
  int idx;
  s_[0] = tv.t[0]+tv.t[1]+tv.t[2];
  s_[1] = tv.t[1]+tv.t[2];
  s_[2] = tv.t[2];
  s_[3] = 0;
  for(idx = 0; idx < 4; idx++)
    if(s_[idx]<min) min = s_[idx];
  for(idx = 0; idx < 4; idx++)
    s_[idx] -= min;
}
TriVect
ShftVect::trivec() const
{
```

```
    TriVect t;
    short mv = -s_[3];
    t.t[2] = s_[2]+mv;
    t.t[1] = s_[1]-t.t[2]+mv;
    t.t[0] = s_[0]-t.t[1]-t.t[2]+mv;
    return t;
}
short
ShftVect::maxshft() const
{
 short mx = s_[0];
 for(int idx=1;idx<=3;idx++)
  if(s_[idx] > mx)
   mx = s_[idx];
 return mx;
}
ShftVect
ShftVect::ralt() const
{
    short a,g,c,t, mn;
    a = s_[0] + short(5.0*drand48()) - 2;
    g = s_[1] + short(5.0*drand48()) - 2;
    c = s_[2] + short(5.0*drand48()) - 2;
    t = s_[3] + short(5.0*drand48()) - 2;
    mn = a;
    if(g<mn) mn = g;
    if(c<mn) mn = c;
    if(t<mn) mn = t;
    ShftVect sv(a-mn,g-mn,c-mn,t-mn);
    return sv;
}
```

```
void
ShftVect::debug( int lvl ) const
{
  ::printf("ShftVect @ %p:",(void*)this);
  ::printf("\ts_: [%hd,%hd,%hd,%hd]\n",s_[0],s_[1],s_[2],s_[3]);
}
ShftVects::ShftVects( int len ) :
 len_(len), svm_(NULL), crit_(NULL), mni_(0), mxi_(len_-1), loops_(0)
{
  svm_ = new ShftVect[ len_ ];
  crit_ = new Criterion[ len_ ];
  if(!svm_ || !crit_) {
   ::fprintf(stderr,"ShftVects::ShftVects(%ld) out of memory.\n",len_);
   exit(1);
   }
  for(int idx = 0; idx < len_; idx++) {
   crit_[idx].idx_ = idx;
   crit_[idx].val_ = 0.0;
   }
}
ShftVects::ShftVects( ShftVects const& rhs ) :
 len_(0), svm_(NULL), crit_(NULL), mni_(0), mxi_(0), loops_(0)
{
 *this = rhs;
}
ShftVects::ShftVects( ShftVect const& sv ) :
 len_(M), svm_(NULL), crit_(NULL), mni_(0), mxi_(M-1), loops_(0)
{
  TriVect t = sv.trivec();
  svm_ = new ShftVect[ len_ ];
  crit_ = new Criterion[ len_ ];
```

```
    if(!svm_ || !crit_) {
     ::fprintf(stderr,"ShftVects::ShftVects(ShftVect&) out of memory.\n");
     exit(1);
     }
    for(int idx = 0; idx < len_; idx++) {
     ShftVect sv( tvm[ idx ] + t );
     svm_[idx] = sv;
     crit_[idx].idx_ = idx;
     crit_[idx].val_ = 0.0;
     }
}
ShftVects::~ShftVects()
{
  if(svm_) { delete [] svm_; svm_ = NULL; }
  if(crit_) { delete [] crit_; crit_ = NULL; }
}
ShftVects const&
ShftVects::operator=( ShftVects const& rhs )
{
  if(this != &rhs) {
   if(svm_) { delete [] svm_; svm_ = NULL; }
   if(crit_) { delete [] crit_; crit_ = NULL; }
   len_ = rhs.len_;
   svm_ = new ShftVect[ len_ ];
   crit_ = new Criterion[ len_ ];
   if(!svm_ || !crit_) {
    ::fprintf(stderr,"ShftVects::operator= out of memory.\n");
    exit(1);
    }
   for(int idx=0; idx<len_; idx++) {
    svm_[idx] = rhs.svm_[idx];
```

```
        crit_[idx] = rhs.crit_[idx];
        }
      mni_ = rhs.mni_;
      mxi_ = rhs.mxi_;
      loops_ = rhs.loops_;
      }
    return *this;
}
void
ShftVects::evaluate( Wvfm const& trace )
{
  for(int idx=mni_;idx<=mxi_;idx++) {
    static int const BGNPT = 299, ENDPT = BGNPT+1399;
    crit_[ idx ].val_ = 0.0;
    crit_[ idx ].idx_ = idx;
    trace.envelope( svm_[idx] );
    if(Wvfm::MAX_INTEGRAL == trace.method())
      for(int sdx=BGNPT; sdx<=ENDPT; sdx++)
        crit_[ idx ].val_ += trace.envv( sdx );
    else
      for(int sdx=BGNPT; sdx<=ENDPT; sdx++)
        crit_[ idx ].val_ += trace.xbnd( sdx );
    }
}
void
ShftVects::debug(int lvl) const
{
  ::printf("ShftVects @ %p\n",(void*)this);
  ::printf("\tlen_ = %ld\n",len_);
  for(int idx = 0; idx < len_; idx++) {
    ::printf("\t%ld ",idx);
```

```
        svm_[idx].debug(lvl);
        crit_[idx].debug(lvl);
        }
      ::printf("\tmxi = %ld  mni = %ld  loops_=%ld\n",mxi_,mni_,loops_);
}
static int
c_cmp( void const* e1, void const* e2 )
{
   double v1 = ((Criterion const*)e1)->val_;
   double v2 = ((Criterion const*)e2)->val_;
   if(v1 > v2)     return 1;
   else if(v1 < v2) return -1;
   else return 0;
}
void
ShftVects::best( int topN )
{
   ShftVects tmpSvs( topN );
   tmpSvs.loops_ = loops_;
   ::qsort( (void*)crit_, len_, sizeof(*crit_), c_cmp );
   for(int tdx=0; tdx<topN; tdx++) {
    int sdx = len_ - topN + tdx;
    int fdx = crit_[ sdx ].idx_;
    tmpSvs.svm_[tdx]  = svm_[ fdx ];
    tmpSvs.crit_[tdx] = crit_[ sdx ];
    }
   *this = tmpSvs;
}
int
ShftVects::terminate()
{
```

```
static int const MAXLOOPS = 100;
double ratio;
mni_ = mxi_ = 0;
for(int idx=1; idx<len_; idx++) {
  double v = crit_[idx].val_;
  if(v >= crit_[mxi_].val_) { mxi_ = idx; }
  if(v <= crit_[mni_].val_) { mni_ = idx; }
  }
svm_[mni_] = svm_[ mxi_ ].ralt();
ratio = crit_[mni_].val_/crit_[mxi_].val_;
return int(((ratio > 0.97) && (loops_ >= 50)) || (loops_ > MAXLOOPS));
}
ShftVect
ShftVects::align( Wvfm const& trace )
{
  while(1) {
   evaluate( trace );
   if(1 == ++loops_)
    best( 25 );
   if(terminate())
    break;
   else
    mxi_ = mni_;
   }
  return svm_[mxi_];
}

#if defined(SA)
int
main(int argc, char const* argv[])
{
```

```
    ShftVect sv, rsv;
    Wvfm wvfm( Wvfm::MIN_XBANDING );
    wvfm.debug();
    rsv = mcalign( wvfm, sv );
    ::printf("The best alignemt is with ");
    rsv.debug();
    return 0;
}
#endif

/*****************************************************************************
 * FILE:        ShftVect.hxx
 * AUTHOR:  Andy Marks
 * Copyright (c) 1995,1996 University of Utah
 */
#ifndef _SHFTVECT_HXX_
#define _SHFTVECT_HXX_
struct TriVect
{
    TriVect();
    TriVect operator+( TriVect const& rhs ) const;
    TriVect( TriVect const& rhs );
    ~TriVect() {};
    void debug( int lvl = 0 ) const;
    short t[3];
};
#if defined(WIN32)
class _declspec( dllexport) ShftVect
#else
class ShftVect
#endif
```

```
{
public:
  ShftVect();
  ShftVect( short v1, short v2, short v3, short v4 );
  ShftVect( TriVect const& tv );
  TriVect trivec() const;
  ShftVect ralt() const;
  void debug( int lvl = 0 ) const;
  short s( int idx ) const { return s_[idx-1]; }
  short maxshft() const;
private:
  short s_[4];
};
class Wvfm;
ShftVect mcalign( Wvfm const& trace, ShftVect const& sv );
#endif

/*****************************************************************************
 * FILE:      spline.cxx
 * TYPIST:    Andy Marks
 *            Human Genetics Dept
 *            Univ of Utah
 * DATE:      Fri Mar 15 12:34:18 MST 1996
 */
#include <stdio.h>
#include <nrc/nr.hxx>
#if !defined(SA)
void
spline( float const x[], float const y[], int n, float yp1, float ypn, float y2[] )
{
 int i, k;
```

```
 float qn,un,*u;
 u = vector(1,n-1);
 if(yp1 > 0.99e30)
  y2[1] = u[1] = 0.0f;
 else {
  y2[1] = -0.5f;
  u[1] = (3.0f/(x[2]-x[1])) * ((y[2]-y[1])/(x[2]-x[1]) - yp1);
  }
 for(i=2; i<=n-1; i++) {
  float p, sig;
  sig = (x[i]-x[i-1]) / (x[i+1]-x[i-1]);
  p = sig*y2[i-1] + 2.0f;
  y2[i]=(sig-1.0f)/p;
  u[i] = (y[i+1]-y[i]) / (x[i+1]-x[i]) - (y[i]-y[i-1]) / (x[i]-x[i-1]);
  u[i] = (6.0f*u[i]/(x[i+1]-x[i-1]) - sig*u[i-1])/p;
  }
 if(ypn > 0.99e30)
  qn = un = 0.0f;
 else {
  qn = 0.5f;
  un = (3.0f/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-1])/(x[n]-x[n-1]));
  }
 y2[n] = (un-qn*u[n-1])/(qn*y2[n-1]+1.0f);
 for(k=n-1; k>=1; k--)
  y2[k] = y2[k]*y2[k+1] + u[k];
 free_vector(u,1,n-1);
}
void
splint(float const xa[], float const ya[], float const y2a[], int n, float x, float *y)
{
 static int klo,khi;
```

```
int k;
float h,b,a;
if(0==klo || 0==khi || x<xa[klo] || x>xa[khi]) {
 klo = 1;
 khi = n;
 while((khi-klo) > 1) {
  k = (khi+klo)>>1;
  if(xa[k]>x) khi = k;
  else klo = k;
  }
 }
h = xa[khi] - xa[klo];
if(0.0f == h) nrerror("Bad xa input to routine splint: the xa's must be distinct");
a = (xa[khi]-x)/h;
b = (x-xa[klo])/h;
*y = a*ya[klo] + b*ya[khi] + ((a*a*a-a)*y2a[klo] + (b*b*b-b)*y2a[khi]) * (h*h)/6.0f;
}
#endif
#if defined(SA)
void
main()
{
 float* x =  vector(1,4);
 float* y =  vector(1,4);
 float* y2 = vector(1,4);
 x[1] = 1.0f; x[2] = 2.0f; x[3] = 3.0f; x[4] = 4.0f;
 y[1] = 1.0f; y[2] = 2.0f; y[3] = 1.0f; y[4] = 2.0f;
 ::spline( x, y, 4, 1.0f, 1.0f, y2 );
 for(float xt = 1.0f; xt < 4.0f; xt += 1.0f/3.0f) {
  float yout;
  ::splint( x, y, y2, 4, xt, &yout );
```

```
    ::printf("xt=%f yout=%f\n",xt,yout);
    }
  }
  #endif

/*************************************************************************
 * FILE:        sw.cxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#include <stdio.h>
#include <limits.h>
#include <string.h>
#include <basecall/sw.hxx>
#include <nrc/nrutil.hxx>
SW::SW( char const* vseq, char const* hseq ) :
 vsz_(0), hsz_(0), outsz_(0), vseq_(NULL), hseq_(NULL),
 score_(INT_MIN), vseqout_(NULL), hseqout_(NULL),
 scores_(NULL), path_(NULL), vcoord_(0), hcoord_(0),
 vpos0_(0), hpos0_(0)
{
  if(!vseq || !hseq) return;
  vsz_ = ::strlen( vseq );
  hsz_ = ::strlen( hseq );
  vseq_ = new char[ 1 + vsz_ + 1 ];
  hseq_ = new char[ 1 + hsz_ + 1 ];
  if(!vseq_ || !hseq_) {
    if(vseq_) delete [] vseq_;
    if(hseq_) delete [] vseq_;
    vseq_ = hseq_ = NULL;
    return;
```

```
    }
    ::strcpy( &vseq_[1], vseq );
    ::strcpy( &hseq_[1], hseq );
    int MVSZ=vsz_+1, MHSZ=hsz_+1;
    scores_ = ::imatrix( 1,MVSZ,  1,MHSZ );
    path_  = ::imatrix( 1,vsz_, 1,hsz_ );
    if(!scores_ || !path_) {
     delete [] vseq_; vseq_ = NULL;
     delete [] hseq_; hseq_ = NULL;
     if(scores_) ::free_imatrix(scores_,1,MVSZ,1,MHSZ); scores_ = NULL;
     if(path_)  ::free_imatrix(path_,  1,vsz_, 1,hsz_ ); path_ = NULL;
     return;
     }
    sw_();
    for(int hdx=2;hdx<=MHSZ;hdx++)
     if(scores_[MVSZ][hdx] > score_) {
      vcoord_ = vsz_;
      hcoord_ = hdx-1;
      score_ = scores_[MVSZ][hdx];
      }
    for(int vdx=2;vdx<=MVSZ;vdx++)
     if(scores_[vdx][MHSZ] > score_) {
      vcoord_ = vdx-1;
      hcoord_ = hsz_;
      score_ = scores_[vdx][MHSZ];
      }
    swwalk_();
}
SW::SW( SW const& rhs ) :
 vsz_(0), hsz_(0), outsz_(0), vseq_(NULL), hseq_(NULL),
 score_(INT_MIN), vseqout_(NULL), hseqout_(NULL),
```

```
 scores_(NULL), path_(NULL), vcoord_(0), hcoord_(0),
 vpos0_(0), hpos0_(0)
{
  *this = rhs;
}
SW const&
SW::operator=( SW const& rhs )
{
 if(this != &rhs) {
  if(vseq_) { delete [] vseq_; vseq_ = NULL; }
  if(hseq_) { delete [] hseq_; hseq_ = NULL; }
  if(vseqout_) { delete [] vseqout_; vseqout_ = NULL; }
  if(hseqout_) { delete [] hseqout_; hseqout_ = NULL; }
  if(scores_) { ::free_imatrix(scores_,1,vsz_+1,1,hsz_+1); scores_=NULL; }
  if(path_)   { ::free_imatrix(path_,1,vsz_,1,hsz_);      path_ =NULL; }
  vsz_ = rhs.vsz_;
  hsz_ = rhs.hsz_;
  outsz_ = rhs.outsz_;
  score_ = rhs.score_;
  vpos0_ = rhs.vpos0_;
  hpos0_ = rhs.hpos0_;
  vseq_ = new char[ vsz_ + 2 ];
  if(vseq_) ::memcpy( vseq_, rhs.vseq_, vsz_+2 );
  hseq_ = new char[ hsz_ + 2 ];
  if(hseq_) ::memcpy( hseq_, rhs.hseq_, hsz_+2 );
  vseqout_ = new char[ outsz_ ];
  if(vseqout_) ::memcpy( vseqout_, rhs.vseqout_, outsz_ );
  hseqout_ = new char[ outsz_ ];
  if(hseqout_) ::memcpy( hseqout_, rhs.hseqout_, outsz_ );
  scores_ = ::imatrix(1,vsz_+1,1,hsz_+1);
  path_  = ::imatrix(1,vsz_, 1,hsz_);
```

```
    if(scores_ && path_) {
     for(int vdx=1;vdx<=vsz_+1;vdx++)
      for(int hdx=1;hdx<=hsz_+1;hdx++) {
       scores_[vdx][hdx] = rhs.scores_[vdx][hdx];
       if(vdx<=vsz_ && hdx<=hsz_)
        path_[vdx][hdx] = rhs.path_[vdx][hdx];
       }
      }
     }
    return *this;
}
SW::~SW()
{
  if(vseq_)   {delete [] vseq_;  vseq_ = NULL; }
  if(hseq_)   {delete [] hseq_;  hseq_ = NULL; }
  if(vseqout_) {delete [] vseqout_;  vseqout_ = NULL; }
  if(hseqout_) {delete [] hseqout_;  hseqout_ = NULL; }
  if(scores_)  {::free_imatrix( scores_,1,vsz_+1,1,hsz_+1);scores_=NULL;}
  if(path_)   {::free_imatrix( path_,1,vsz_,1,hsz_ );path_ = NULL;}
}
void
SW::sw_() const
{
  static int const MATCHVAL =  1, MISMATCHVAL = -1,
                VGAP_VAL = -3, HGAP_VAL   = -3;
  int vdx, hdx;
  for(vdx=1;vdx<=vsz_+1;vdx++)
   for(hdx=1;hdx<=hsz_+1;hdx++) {
    scores_[vdx][hdx] = 0;
    if(vdx<=vsz_ && hdx<=hsz_)
     path_[vdx][hdx] = 0;
```

```
    }
  for(int i=1;i<=vsz_;i++)
   for(int j=1;j<=hsz_;j++) {
    int val[4], I, m = (vseq_[i]==hseq_[j])? MATCHVAL: MISMATCHVAL;
    val[1] = scores_[i][j+1] + VGAP_VAL;
    val[2] = scores_[i][j]   + m;
    val[3] = scores_[i+1][j] + HGAP_VAL;
    int mx = val[I=1];
    for(int k=2;k<=3;k++)
     if(val[k]>mx)
      mx = val[I=k];
    scores_[i+1][j+1] = mx;
    path_[i][j] = I-2;
    }
}
void
SW::swwalk_()
{
  static char const GAPCHAR = '-';
  int idx, N, vpos, hpos, pos;
  char *v, *h;
  N = vcoord_+hcoord_;
  v = new char[ 1+N+1 ]; if(!v) return;
  h = new char[ 1+N+1 ]; if(!h) { delete [] v; return; }
  pos = N;
  vpos = vcoord_;
  hpos = hcoord_;
  for(idx=0;idx<=(N+1);idx++)
   v[idx] = h[idx] = 0;
  while(vpos>0 && hpos>0) {
   switch( path_[vpos][hpos] ) {
```

```
            case -1: v[pos] = vseq_[ vpos-- ]; h[pos] = GAPCHAR;      break;
            case 0: v[pos] = vseq_[ vpos-- ]; h[pos] = hseq_[ hpos-- ]; break;
            case 1: v[pos] = GAPCHAR;       h[pos] = hseq_[ hpos-- ]; break;
            default:
              ::fprintf(stderr,"SW::swwalk_() encountered %d, not in [-1,..,1]\n",
                path_[vpos][hpos]);
              delete [] v;
              delete [] h;
              return;
            }
          pos--;
          }
        pos++;
        vpos0_ = vpos;
        hpos0_ = hpos;
        if(vseqout_) { delete [] vseqout_; vseqout_ = NULL; }
        if(hseqout_) { delete [] hseqout_; hseqout_ = NULL; }
        outsz_ = N-pos+1+1;
        vseqout_ = new char[ outsz_ ];
        hseqout_ = new char[ outsz_ ];
        if(vseqout_) ::strcpy( vseqout_, &v[pos] );
        if(hseqout_) ::strcpy( hseqout_, &h[pos] );
        delete [] v;
        delete [] h;
    }
    void
    SW::debug() const
    {
      ::printf("SW @ %p\n",(void*)this);
      ::printf("  vsz_ = %2d  hsz_=%2d outsz_=%2d\n",vsz_,hsz_,outsz_);
      ::printf("  vseq_ = [%s]\n",vseq_?&vseq_[1]:"null");
```

```
        ::printf("  hseq_ = [%s]\n",hseq_?&hseq_[1]:"null");
        ::printf("  vpos0_ = %2d  hpos0_ = %2d\n",vpos0_,hpos0_);
        ::printf("  score_=%2d\n",score_);
        ::printf("  vcoord_ = %4d  hcoord_ = %4d\n",vcoord_,hcoord_);
        ::printf("  vseqout_ = [%s]\n",vseqout_?vseqout_:"null");
        ::printf("  hseqout_ = [%s]\n",hseqout_?hseqout_:"null");
        ::printf("scores_:\n");
        if(scores_)
         for(int vdx=1;vdx<=vsz_+1;vdx++) {
          ::printf("  ");
          for(int hdx=1;hdx<=hsz_+1;hdx++)
           ::printf("%2d ".scores_[vdx][hdx]);
          ::printf("\n");
          }
        ::printf("path_:\n");
        if(path_)
         for(int vdx=1;vdx<=vsz_;vdx++) {
          ::printf("  ");
          for(int hdx=1;hdx<=hsz_;hdx++)
           ::printf("%2d ",path_[vdx][hdx]);
          ::printf("\n");
          }
      }
      #if defined(SA)
      int
      main()
      {
        char const* transposon = "tccattggccctcaaacccc";
        char const* observed   = "tccattggccctccaaacccc";
        SW sw( transposon, observed );
        sw.debug();
```

```
    return 0;
}
#endif

/*************************************************************************
 * FILE:        sw.hxx
 * AUTHOR:  Andy Marks
 * COPYRIGHT (c) 1996, University of Utah
 */
#if !defined(_SW_HXX_)
#define _SW_HXX_
#if defined(WIN32)
class _declspec( dllexport ) SW
#else
class SW
#endif
{
public:
  SW( char const* known, char const* unknown );
  SW( SW const& rhs );
  SW const& operator=(SW const& rhs);
  ~SW();
  int score()      const { return score_; }
  int vcoord()     const { return vcoord_; }
  int hcoord()     const { return hcoord_; }
  int vpos0()      const { return vpos0_; }
  int hpos0()      const { return hpos0_; }
  char const* vout() const { return vseqout_; }
  char const* hout() const { return hseqout_; }
  void debug() const;
private:
```

```
    int vsz_,
        hsz_,
        outsz_,
        vpos0_,
        hpos0_;
    char *vseq_,
        *hseq_;
    int score_;
    char *vseqout_,
        *hseqout_;
    int **scores_,
        **path_;
    int vcoord_,
        hcoord_;
    void sw_() const;
    void swwalk_();
};
#endif

/*************************************************************
 * FILE:        Wvfm.cxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
#include <nrc/nr.hxx>
#include <nrc/Complex.hxx>
#include <time.h>
#include <basecall/RatioBin.hxx>
Wvfm::Wvfm() :
  rows_(0). cols_(0), bgni_(0), endi_(0), ds_(UNKNOWN), pm_(NULL),
```

```
 pv_(NULL), pi_(NULL), px_(NULL), pb_(NULL), method_(MAX_INTEGRAL),
 status_(STS_UNINITD), ib_(_2X), fCalcSST_(1), obgni_(0), oendi_(0)
{
  lnordr_[0] = '\0';
  for(int idx=0;idx<4;idx++)
   for(int jdx=0;jdx<4;jdx++)
    ssm_[idx][jdx] = (idx==jdx)? 1.0: 0.0;
  status_ = STS_NO_SIZE;
}
Wvfm::Wvfm( char const* dataf, char const* parmf, char const* lnordr, DATASRC dsrc,
Method m ) :
 rows_(0), cols_(0), bgni_(0), endi_(0), ds_(dsrc), pm_(NULL),
 pv_(NULL), pi_(NULL), px_(NULL), pb_(NULL), method_(m),
 status_(STS_UNINITD), ib_(_2X), fCalcSST_(1), obgni_(0), oendi_(0)
{
  int BBUFSZ;
  int** bbuf, rdx, cdx, scanl, idx, jdx;
  char scratch[ 128 ];
  FILE* fp = NULL;
  for(idx=0;idx<4;idx++)
   for(jdx=0;jdx<4;jdx++)
    ssm_[idx][jdx] = (idx==jdx)? 1.0: 0.0;
  if(!parmf) {
   if(lnordr) {
    ::strcpy(lnordr_,lnordr);
    lnordr_[4] = '-';
    lnordr_[5] = '\0';
    }
   else {
    status_ = STS_NO_LNORD;
    return;
```

```
    }
   }
  else if(NULL==(fp=::fopen(parmf,"r"))) {
   status_ = STS_NO_PARMF;
   return;
   }
  else {
   if(NULL != lnordr)
    ::strcpy(lnordr_,lnordr);
   else {
    ::fscanf( fp, "%s\n", scratch );
    ::fscanf( fp, "%s\n", lnordr_);
    ::fscanf( fp, "%s\n", scratch );
    }
   lnordr_[4] = '-';
   lnordr_[5] = '\0';
   for(idx = 0; idx < 4; idx++)
    if(4 != ::fscanf(fp,"%lf %lf %lf %lf",
     &ssm_[idx][0], &ssm_[idx][1], &ssm_[idx][2], &ssm_[idx][3] )) {
     ::fprintf(stderr,"SSM file[%s] has [%s] instead of 4 doubles.\n",
      parmf,scratch);
     ::fprintf(stderr,"Wrong file, or incorrect file format.\n");
     exit(1);
     }
   ::fclose(fp);
   fCalcSST_ = 0;
   }
  if(!dataf || (NULL == (fp = ::fopen(dataf,"r")))) {
   status_ = STS_NO_DATAF;
   return;
   }
```

```
#undef OLD_FMT
#if defined(OLD_FMT)
  BBUFSZ = 12000;
  while(1) {
   if(NULL == ::fgets( scratch, sizeof(scratch), fp))
    break;
   else if(0 == ::strncmp(scratch,"INTENSITY_DATA",14))
    break;
   }
#else
  char str[2][64];
  int nfluor;
  float fs;
  int dummy;
  int NF;
  NF = ::fscanf(fp,"%s %s %d %d %f
%d\n",str[0],str[1],&nfluor,&BBUFSZ,&fs,&dummy);
  if(4 != nfluor) {
   ::fprintf(stderr,"%s:%d, $3==%d, not 4\n",__FILE__,__LINE__,nfluor);
   status_ = STS_IFILE_UN;
   return;
   }
  if(1.75f != fs && 2.00f != fs) {
   ::fprintf(stderr,"%s:%d, $4==%4.2f, neither 1.75 nor 2.00\n",__FILE__,__LINE__,fs);
   status_ = STS_IFILE_UN;
   return;
   }
#endif
  if(NULL == (bbuf = ::imatrix( 1,BBUFSZ, 1,4 ))) {
   status_ = STS_NO_MEM;
   return;
```

```
  }
 for(scanl=1; scanl<=BBUFSZ; scanl++)
  if(NULL == ::fgets( scratch, sizeof(scratch), fp ))
   break;
  else if(0 == ::strncmp(scratch,"INTENSITY_DATA_END:",19))
   break;
  else {
   float b[4];
   int* p = bbuf[scanl];
   if(4 != ::sscanf( scratch, "%f %f %f %f",&b[0],&b[1],&b[2],&b[3])) {
    ::free_imatrix( bbuf, 1,BBUFSZ,1,4 );
    status_ = STS_SCNL_RD_FAIL;
    return;
    }
   else {
    p[1] = int( b[0]+0.5f );
    p[2] = int( b[1]+0.5f );
    p[3] = int( b[2]+0.5f );
    p[4] = int( b[3]+0.5f );
    }
   }
 ::fclose(fp);
 scanl--;
 if(NULL==(pm_=::dmatrix(1,rows_=scanl,1,cols_=4))) {
  ::free_imatrix( bbuf,1,BBUFSZ,1,4 );
  status_ = STS_NO_MEM;
  return;
  }
 else for(rdx=1;rdx<=rows_;rdx++)
  for(cdx=1;cdx<=cols_;cdx++)
   pm_[rdx][cdx] = double( bbuf[rdx][cdx] );
```

```
  ::free_imatrix( bbuf, 1,BBUFSZ, 1,4 );
  status_ = STS_INITD;
}
Wvfm::Wvfm( int r, int c, char const* lnordr, DATASRC ds, Method m ) :
 rows_(r), cols_(c), bgni_(1), endi_(0), ds_(ds), pm_(NULL),
 pv_(NULL), pi_(NULL), px_(NULL), pb_(NULL), method_(m),
 status_(STS_UNINITD), ib_(_2X), fCalcSST_(1), obgni_(1), oendi_(0)
{
  int idx, jdx, cdx, rdx;
  if(0 == endi_) endi_ = scanl();
  lnordr_[0] = '\0';
  if(lnordr)
   ::strcpy( lnordr_, lnordr );
  pm_ = ::dmatrix( 1,rows_, 1,cols_);
  for(cdx=1; cdx<=cols_; cdx++)
   for(rdx=1; rdx<=rows_; rdx++)
    pm_[rdx][cdx] = 0.0;
  for(idx=0;idx<4;idx++)
   for(jdx=0;jdx<4;jdx++)
    ssm_[idx][jdx] = (idx==jdx)? 1.0: 0.0;
  status_ = STS_INITD;
}
Wvfm::Wvfm( Wvfm const& rhs ) :
 rows_(0), cols_(0), bgni_(0), endi_(0), pm_(NULL),
 pv_(NULL), pi_(NULL), px_(NULL), pb_(NULL), method_(MAX_INTEGRAL),
 ib_(_2X), fCalcSST_(1), obgni_(0), oendi_(0)
{
  *this = rhs;
}
Wvfm const&
Wvfm::operator=( Wvfm const& rhs)
```

```
{
 if(this != &rhs) {
  int NS, idx, jdx;
  release();
  status_ = rhs.status_;
  rows_   = rhs.rows_;
  cols_   = rhs.cols_;
  obgni_  = rhs.obgni_;
  oendi_  = rhs.oendi_;
  bgni_   = rhs.bgni_;
  endi_   = rhs.endi_;
  ds_     = rhs.ds_;
  method_ = rhs.method_;
  ib_     = rhs.ib_;
  fCalcSST_ = rhs.fCalcSST_;
  if(rhs.pm_ && 0!=rows_ && 0!=cols_)
   if(NULL == (pm_ = ::dmatrix(1,rows_,1,cols_))) {
    status_ = STS_NO_MEM;
    goto bugout;
    }
   else for(idx=1;idx<=rows_;idx++)
    for(int jdx=1;jdx<=cols_;jdx++)
     pm_[idx][jdx] = rhs.pm_[idx][jdx];
  ::strcpy(lnordr_,rhs.lnordr_);
  for(idx=0;idx<4;idx++)
   for(jdx=0;jdx<4;jdx++)
    ssm_[idx][jdx] = rhs.ssm_[idx][jdx];
  NS = scanl();
  if(rhs.pv_ && rhs.pi_ && rhs.px_ && rhs.pb_) {
   pv_ = ::dvector( 1, NS );
   px_ = ::dvector( 1, NS );
```

```
        pi_ = ::ivector( 1, NS );
        pb_ = ::vector(  1, NS );
        if(!pv_ || !pi_ || !px_ || !pb_) {
         status_ = STS_NO_MEM;
         goto bugout;
         }
        for(idx=1;idx<=NS;idx++) {
         pv_[idx] = rhs.pv_[idx];
         pi_[idx] = rhs.pi_[idx];
         px_[idx] = rhs.px_[idx];
         pb_[idx] = rhs.pb_[idx];
         }
        }
       }
bugout:
   return *this;
}
void
Wvfm::release()
{
   int NS = scanl();
   if(NULL != pm_) { ::free_dmatrix( pm_,1,rows_,1,cols_); pm_ = NULL; }
   if(NULL != pv_) { ::free_dvector( pv_, 1, NS ); pv_ = NULL; }
   if(NULL != pi_) { ::free_ivector( pi_, 1, NS ); pi_ = NULL; }
   if(NULL != px_) { ::free_dvector( px_, 1, NS ); px_ = NULL; }
   if(NULL != pb_) { ::free_vector(  pb_, 1, NS ); pb_ = NULL; }
}
Wvfm::~Wvfm()
{
   release();
}
```

```
struct VI { double v; int i,z ; };
static int
vi_cmp(void const* p1, void const* p2)
{
  double v1 = ((VI const*)p1)->v;
  double v2 = ((VI const*)p2)->v;
  if(v1>v2) return 1;
  else if(v1<v2) return -1;
  else return 0;
}
void
Wvfm::bgnEnd()
{
  if(bgni_>=2 && endi_>bgni_ && endi_<scanl()) {
   obgni_ = bgni_;
   oendi_ = endi_;
   return;
   }
  static int const NZ = 8;
  VI vi[1+NZ];
  int ZSZ = scanl()/NZ;
  for(int nz=1;nz<=NZ;nz++) {
   int zb = 1+(nz-1)*ZSZ, ze = nz*ZSZ, mi;
   double mv = sc_la(mi=zb,1);
   for(int s=zb+1;s<=ze;s++) {
    for(int f=2;f<=lanes();f++) {
     double v = sc_la(s,f);
     if(v>mv) { mv=v; mi=s; }
     }
    }
   vi[nz].v = mv;
```

```
 vi[nz].i = mi;
 vi[nz].z = nz;
 }
::qsort(&vi[1],NZ,sizeof(VI),vi_cmp);
int mx1 = vi[NZ].z;
int mx2 = vi[NZ-1].z;
double primerPeak;
if(1==abs(mx1-mx2)) {
 bgni_ = vi[NZ].i;
 primerPeak = vi[NZ].v;
 endi_ = scanl()-100;
 }
else if(mx1<mx2) {
 bgni_ = vi[NZ].i;
 primerPeak = vi[NZ].v;
 endi_ = vi[NZ-1].i-100;
 }
else {
 bgni_ = vi[NZ-1].i;
 primerPeak = vi[NZ-1].v;
 endi_ = vi[NZ].i-100;
 }
int B=bgni_;
for(int ns=bgni_+30;ns<=bgni_+400;ns++)
 for(int f=1;f<=lanes();f++)
  if(sc_la(ns,f) > 0.3*primerPeak) {
   if(bgni_==B) B = ns;
   continue;
   }
double mu=0.0, nv=0.0;
int m, f. LHS = (bgni_+endi_)/2;
```

```
      for(m=bgni_;m<=LHS;m++) {
       double lmx = sc_la(m,1);
       for(f=2;f<=lanes();f++)
        if(sc_la(m,f) > lmx)
          lmx = sc_la(m,f);
       mu += lmx;
       nv += 1.0;
       }
      mu /= nv;
      nv = 0.0;
      for(m=bgni_;m<=LHS;m++) {
       double lmx = sc_la(m,1);
       for(f=2;f<=lanes();f++)
        if(sc_la(m,f) > lmx)
          lmx = sc_la(m,f);
       if(0.0==nv && lmx<=mu)
        nv++;
       else if(0.0!=nv && lmx>mu) {
        bgni_ = m;
        break;
        }
       }
    #if 1
      endi_ -= 350;
    #else
      double denom, numer, nb;
      numer = double(endi_-bgni_+1);
      if(_1X==ib_) denom = 10.0;
      else if(_2X==ib_) denom = 7.0;
      else denom = 5.0;
      nb = numer/denom;
```

```
    if(nb>=650.0) endi_ -= int((nb-650.0)*denom);
#endif
    obgni_ = bgni_;
    oendi_ = endi_;
}
static void
fsmPkdet(double const* vec,short b,short e,short& P1,short& PN,short* pk,short* tr)
{
    enum STATE { ST_UK, ST_UP, ST_DN } st = ST_UK;
    short scnl,TN;
    double vm1;
    TN = PN = 0;
    vm1 = vec[ scnl=b ];
    for(++scnl; scnl<=e; scnl++) {
     double v = vec[scnl];
     switch(st) {
      case ST_UK:
       if(v > vm1)     st=ST_UP;
       else if(v < vm1) st=ST_DN;
       break;
      case ST_UP:
       if(v < vm1) { st = ST_DN; pk[ ++PN ] = scnl-1; }
       break;
      case ST_DN:
       if(v > vm1) { st = ST_UP; tr[ ++TN ] = scnl-1; }
       break;
      }
     vm1 = v;
     }
    P1=1;
    if(pk[P1] < tr[1]) P1++;
```

```
    if(pk[PN] > tr[TN]) PN--;
  }
  static void
  bslAdjust( double* bsl, int bgn, int end )
  {
    static int const LW = 125;
    int N, FFTSZ;
    double* minv, *hn, *lpfbsl, *rawmin=&bsl[bgn-1];
  #  define REAL(n) (2*(n)-1)
  #  define IMAG(n) (2*(n))
    int c;
    N = end-bgn+1;
    FFTSZ = 1 << int(::ceil(::log(3*N+(2*LW-1)-1)/::log(2.0)) );
    minv = ::dvector(1,2*FFTSZ);
    hn   = ::dvector(1,2*FFTSZ);
    lpfbsl = ::dvector(1,N);
    for(c=1;c<=2*FFTSZ;c++)
     minv[c] = hn[c] = 0.0;
    for(c=1;c<=N;c++) {
     double v = rawmin[c];
     minv[REAL(N+1-c)]   = v;
     minv[REAL(N+c)]     = v;
     minv[REAL(3*N+1-c)] = v;
     }
    ::dfour1( minv, FFTSZ, 1 );
    for(c=1;c<=LW;c++) {
     double v = 0.5*(1.0 + ::cos(double(c-1)*NR_PI/double(LW)))/double(LW);
     hn[REAL(c)] = v;
     if(1!=c) hn[REAL(FFTSZ-c+2)] = v;
     }
    ::dfour1( hn, FFTSZ, 1 );
```

```
    ::dCMul( minv, hn, hn, FFTSZ );
    ::dfour1( hn, FFTSZ, -1 );
    for(c=1;c<=N;c++)
     lpfbsl[c] = hn[REAL(N+c)]/double(FFTSZ);
    double BX=0.0, BY=0.0, BN=0.0, EX=0.0, EY=0.0, EN=0.0;
    double sl, in;
    for(c=1;c<=N/10;c++) {
     double bdel=lpfbsl[c]-rawmin[c],
          edel=lpfbsl[N+1-c]-rawmin[N+1-c];
     if(bdel>0.0) { BN++; BX+=double(c);    BY+=bdel; }
     if(edel>0.0) { EN++; EX+=double(N+1-c); EY+=edel; }
     }
    BX /= BN; BY /= BN;
    EX /= EN; EY /= EN;
    sl = (EY-BY)/(EX-BX);
    in = EY - sl*EX;
    for(c=1;c<=N;c++)
     rawmin[c] = lpfbsl[c] - (double(c)*sl+in);
    ::free_dvector(minv,1,2*FFTSZ);
    ::free_dvector(hn, 1,2*FFTSZ);
    ::free_dvector(lpfbsl, 1,N);
}
void
Wvfm::fbbls_(int fnr, double* bsl, DIRECTION_ DIR) const
{
#undef DOC_PATENT
#if defined(DOC_PATENT)
 static int sl=1;
#endif
   double TAU=75.0;
   double prvmin, newmin, K, cnt=1.0, sl, in, thr, dy,dx;
```

```
int lstknt,x, IC,FC,LC;
 if(FWD==DIR) {
  lstknt=bgni_; IC=bgni_+1; FC=endi_+1;
  }
 else {
  lstknt=endi_; IC=endi_-1; FC=bgni_-1;
  }
prvmin = sc_la(lstknt,fnr);
LC = IC+10*DIR;
for(x=IC;x!=LC;x+=DIR)
  if(0.0 != (prvmin=sc_la(x,fnr)))
    break;
K = (prvmin/3.0)/(exp(1.0)-1.0);
for(int snr=IC;snr!=FC;snr+=DIR) {
  thr = prvmin + K*(exp(++cnt/TAU)-1.0);
  if(0.0==(newmin=sc_la(snr,fnr)))
    newmin = thr+1.0;
  if(newmin<=thr || cnt>=100.0) {
    if(cnt>=100.0 && thr<newmin)
      newmin = thr;
    dy = newmin-prvmin;
    dx = double(snr-lstknt);
    sl = dy/dx;
    in = newmin - sl*snr;
    prvmin = newmin;
    for(x=lstknt;x!=snr;x+=DIR) {
      double v = double(x)*sl+in;
      bsl[x] = (FWD==DIR)? v: sqrt(v*bsl[x]);
#if defined(DOC_PATENT)
if(3==fnr && 1==s1) ::printf("%d %7.2f %7.2f %7.2f\n",DIR,sc_la(x,fnr),v,bsl[x]);
#endif
```

```
        }
        lstknt = snr;
        cnt = 1.0;
        K = (prvmin/3.0)/(exp(1.0)-1.0);
        }
      }
     snr -= DIR;
     if(snr==lstknt) {
      double v = double(snr)*sl+in;
      bsl[snr] = (FWD==DIR)? v: sqrt(v*bsl[snr]);
#if defined(DOC_PATENT)
if(3==fnr && 1==s1) ::printf("%d %7.2f %7.2f %7.2f\n",DIR,sc_la(snr,fnr),v,bsl[snr]);
#endif
     }
     else {
      if(thr<newmin) newmin = thr;
      dy = newmin-prvmin;
      dx = double(snr-lstknt);
      sl = dy/dx;
      in = newmin - sl*snr;
      LC = snr+DIR;
      for(x=lstknt;x!=LC;x+=DIR) {
       double v = double(x)*sl+in;
       bsl[x] = (FWD==DIR)? v: sqrt(v*bsl[x]);
#if defined(DOC_PATENT)
if(3==fnr && 1==s1) ::printf("%d %7.2f %7.2f %7.2f\n",DIR,sc_la(x,fnr),v,bsl[x]);
#endif
      }
     }
#if defined(DOC_PATENT)
  if(3==fnr && FWD!=DIR) { s1 = 0; exit(1); }
```

```
#endif
}
void
Wvfm::bestBaseline_()
{
  double *bsl = ::dvector(1,endi_);
  for(int fnr=1;fnr<=lanes();fnr++) {
   fbbls_( fnr, bsl, FWD );
   fbbls_( fnr, bsl, BCK );
   bslAdjust( bsl, bgni_, endi_ );
   for(int snr=bgni_;snr<=endi_;snr++) {
    double del = sc_la(snr,fnr)-bsl[snr];
    if(del<0.0) del = 0.0;
    sc_la_set(snr,fnr, del);
   }
  }
#if 0
  debug("AFTER BESTBASLINE");
#endif
  ::free_dvector(bsl,1,endi_);
}
int
Wvfm::rawPks_( RatioBin& rb ) const
{
  short* ppk = new short[(endi_-bgni_+1)/2];
  short* ptr = new short[(endi_-bgni_+1)/2];
  double* penv = new double[1+endi_];
  short lane, scnl, P1, PN;
  if(!ppk || !ptr || !penv)
   return 0;
  for(scnl=bgni_;scnl<=endi_;scnl++) {
```

```
      penv[scnl] = sc_la(scnl,1);
      for(lane=2;lane<=lanes();lane++)
       if(sc_la(scnl,lane)>penv[scnl])
        penv[scnl] = sc_la(scnl,lane);
      }
    fsmPkdet( penv, bgni_, endi_, P1, PN, ppk, ptr);
    if(0==PN)
     return 0;
    for(short idx=P1; idx<=PN; idx++) {
     double colsmpl[4];
     short x = ppk[idx];
     for(lane=1;lane<=lanes();lane++)
      colsmpl[lane-1] = sc_la(x,lane);
     rb.classify( colsmpl, idx );
     }
    delete [] ppk;
    delete [] ptr;
    delete [] penv;
    return 1;
}
int
Wvfm::specSep()
{
  int scnl, lane, mrow;
  if(0==bgni_ || 0==endi_ || bgni_>endi_ || 0==rows_ || 0==cols_) {
   status_ = STS_CORRUPT;
   return 0;
   }
#if 0
  debug("Before bestBaseline_");
#endif
```

```
  bestBaseline_();
#if 0
  debug("After bestBaseline_");
#endif
  if(fCalcSST_) {
   RatioBin b( lanes() );
   Wvfm tmp = *this;
   if(1!=tmp.rawPks_( b )) {
    status_ = STS_CORRUPT;
    return 0;
    }
   if(1 != b.analyze()) {
    status_ = STS_CORRUPT;
    return 0;
    }
   for(short row=1;row<=lanes();row++)
    for(short col=1;col<=lanes();col++)
     ssm_[row-1][col-1] = b.sst(row,col);
   }
  double minv[5];
  minv[1] = minv[2] = minv[3] = minv[4] = 0.0;
  for(scnl=bgni_;scnl<=endi_;scnl++) {
   double temp[4];
   for(mrow=0;mrow<lanes();mrow++) {
    double sum=0.0;
    for(lane=1; lane<=lanes(); lane++)
     sum += sc_la(scnl,lane) * ssm_[mrow][lane-1];
    temp[mrow] = sum;
    if(sum<minv[mrow+1]) minv[mrow+1] = sum;
    }
   for(lane=1;lane<=lanes();lane++)
```

```
        sc_la_set(scnl,lane,temp[lane-1]);
      }
     for(scnl=bgni_;scnl<=endi_;scnl++)
      for(lane=1;lane<=lanes();lane++) {
       double v = sc_la(scnl,lane);
#if 1
      if(v<0.0) sc_la_set(scnl,lane,0.0);
#else
      if(v<0.0) sc_la_set(scnl,lane, v / -minv[lane] );
#endif
      }
#if 0
  debug("After SST applied");
#endif
  return 1;
}
void
Wvfm::sort()
{
  int transposed = 0;

  if(rows_ > cols_) {
   transpose();
   transposed = 1;
   }
  for(int ldx=1; ldx <= rows_; ldx++)
   ::qsort( &pm_[ldx][1], cols_, sizeof(double), d_cmp );
  if(1 == transposed)
   transpose();
}
static void
```

```
bubbleswap(double* v, int* i, int i1, int i2)
{
  double td;
  int   ti;
  td   = v[i1]; ti   = i[i1];
  v[i1] = v[i2]; i[i1] = i[i2];
  v[i2] = td;   i[i2] = ti;
}
void
Wvfm::envelope( ShftVect const& sv ) const
{
  static float const MAXXBND = 2.0f;
  static float const MINXBND = 1.0f;
  Wvfm* lt = (Wvfm*)this;
  int scnl. SVMAX = 1+sv.maxshft();
  int  mi. NS = scanl(), NL = lanes();

  if(!pv_) lt->pv_ = ::dvector( 1, NS );
  if(!pi_) lt->pi_ = ::ivector( 1, NS );
  if(!px_) lt->px_ = ::dvector( 1, NS );
  if(!pb_) lt->pb_ = ::vector( 1, NS );
  if(!pv_ || !pi_ || !px_ || !pb_) {
   lt->status_ = STS_NO_MEM;
   return;
   }
  for(scnl=1; scnl<SVMAX; scnl++) {
   lt->pv_[scnl] = lt->px_[scnl] = 0.0;
   lt->pb_[scnl] = 0.0f;
   lt->pi_[scnl] = 0;
   }
  for(scnl=SVMAX; scnl<=NS; scnl++) {
```

```
        double mx, smx, v[5];
        int i[5];
        v[1] = sc_la( scnl-sv.s(1), 1 ); i[1]=1;
        v[2] = sc_la( scnl-sv.s(2), 2 ); i[2]=2;
        v[3] = sc_la( scnl-sv.s(3), 3 ); i[3]=3;
        v[4] = sc_la( scnl-sv.s(4), 4 ); i[4]=4;
        if(v[1]>v[2]) bubbleswap(v,i,1,2);
        if(v[2]>v[3]) bubbleswap(v,i,2,3);
        if(v[3]>v[4]) bubbleswap(v,i,3,4);
        if(v[1]>v[2]) bubbleswap(v,i,1,2);
        if(v[2]>v[3]) bubbleswap(v,i,2,3);
        if(v[1]>v[2]) bubbleswap(v,i,1,2);
        mx = v[4]; smx = v[3]; mi=i[4];
        lt->pv_[ scnl ] = mx;
        lt->pi_[ scnl ] = mi;
        if(v[1] < 0.0) v[1] = 0.0;
        if(v[3] < 0.0) v[3] = 0.0;
        lt->pb_[ scnl ] = (v[4]!=v[3])? float((v[3]-v[1])/(v[4]-v[3])): 0.5f;
        if(lt->pb_[ scnl ] > 9.99f) lt->pb_[scnl] = 9.99f;
        if(smx < DBL_EPSILON) smx = DBL_EPSILON;
        if(mx < 0.1)
          mx = MINXBND + mx/sqrt(smx);
        else
          mx /= smx;
        if(mx > MAXXBND)     mx = MAXXBND;
        else if(mx < MINXBND) mx = MINXBND;
        lt->px_[ scnl ] = mx;
        }
    }
    void
    Wvfm::transpose()
```

```
{
  Wvfm tmp( cols_, rows_, lnordr_ );
  double **td;
  for(int rdx=1;rdx<=rows_;rdx++)
    for(int cdx=1;cdx<=cols_;cdx++)
      tmp[cdx][rdx] = pm_[rdx][cdx];
  rows_ = tmp.rows_;
  cols_ = tmp.cols_;
  td = tmp.pm_;
  tmp.pm_ = pm_;
  pm_ = td;
  tmp.rows_ = cols_;
  tmp.cols_ = rows_;
}
void
Wvfm::append( Wvfm const& rhs, int oscnl, int nscnl )
{
  int NS = (rhs.endi()<rhs.scanl())? rhs.endi(): rhs.scanl();
  Wvfm tmp( oscnl+(NS-nscnl+1), 4, lnordr_ );
  ShftVect noshift;
  int or=rows_, oc=cols_;
  double **td = tmp.pm_;
  for(int lane=1;lane<=4;lane++) {
    for(int os=1;os<=oscnl;os++)
      tmp.sc_la_set( os, lane, sc_la(os,lane) );
    for(int ns=nscnl;ns<=NS;os++,ns++)
      tmp.sc_la_set( os, lane, rhs.sc_la(ns,lane) );
    }
  rows_ = tmp.rows_;
  cols_ = tmp.cols_;
  tmp.pm_ = pm_;
```

```
    pm_ = td;
    tmp.rows_ = or;
    tmp.cols_ = oc;
    bgni_ = tmp.bgni_;
    endi_ = tmp.endi_;
    NS = scanl();
    if(NULL != pv_) { ::free_dvector( pv_, 1, NS ); pv_ = NULL; }
    if(NULL != pi_) { ::free_ivector( pi_, 1, NS ); pi_ = NULL; }
    if(NULL != px_) { ::free_dvector( px_, 1, NS ); px_ = NULL; }
    if(NULL != pb_) { ::free_vector( pb_, 1, NS ); pb_ = NULL; }
    envelope( noshift );
}
void
Wvfm::lnordr( char const* lnordr )
{
  for(int idx=0;idx<6;idx++)
    lnordr_[idx] = lnordr[idx];
}
void
Wvfm::ssm( double const* pssm )
{
  fCalcSST_ = 0;
  for(int idx=0; idx<4; idx++)
    for(int jdx=0; jdx<4; jdx++)
      ssm_[idx][jdx] = pssm[idx*4+jdx];
}
void
Wvfm::pm( double** pn, int r, int b, int e )
{
  release();
  pm_ = pn;
```

```
    rows_ = r;
    bgni_ = b;
    endi_ = (e>rows_)? rows_: e;
}
void
Wvfm::debug( char const* msg ) const
{
    static char const* METLUT[] =
    {
     "MAX_INTEGRAL", "MIN_XBANDING"
    };
    static char const* DSLUT[] =
    {
     "UNKNOWN", "ABI ", "MDYN", "FLUOR", "TRUVEL"
    };
    ::printf("Wvfm @ %p\n",(void*)this);
    if(NULL != msg)
     ::printf("  %s\n",msg);
    ::printf("\trows=%u cols=%u bgni=%u endi=%u method=%s\n",
     rows_,cols_,bgni_,endi_,  METLUT[method()] );
    ::printf("datasrc: [%s]\n", DSLUT[ds_] );
    ::printf("lnordr:  [%s]\n",('\0'==lnordr_[0]? "": lnordr_));
    if(ABI==ds_ || MDYN==ds_) {
     ::printf("specsep:\n");
     for(int idx=0;idx<4;idx++) {
      ::printf("  ");
      for(int jdx=0;jdx<4;jdx++)
       ::printf("%9.6f ",ssm_[idx][jdx]);
      ::printf("\n");
     }
    }
```

```
    if(NULL == pm_)
     ::printf("  pm = (null)\n");
    else {
     int idx;
     ::printf("  pm @ %p\n",(void*)pm_);
     if(rows_>cols_)
      for(idx = bgni_; idx <= endi_; idx++) {
       ::printf("%4ld %11.6lf %11.6lf %11.6lf %11.6lf\n",
        idx,pm_[idx][1],pm_[idx][2],pm_[idx][3],pm_[idx][4]);
       }
     else for(idx = bgni_; idx <= endi_; idx++) {
      ::printf("%4ld %11.6lf %11.6lf %11.6lf %11.6lf\n",
       idx,pm_[1][idx],pm_[2][idx],pm_[3][idx],pm_[4][idx]);
      }
    }
}

/*************************************************************
 * FILE:        Wvfm.hxx
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#ifndef _WVFM_HXX_
#define _WVFM_HXX_
#include <basecall/ShftVect.hxx>
class RdrOut;
class SegRead;
class RatioBin;
#if defined(WIN32)
class _declspec (dllexport)  Wvfm
#else
```

```
class Wvfm
#endif
{
public:
 enum INTERPOLATE_BY { _3X, _2X, _1X };
 enum DATASRC { UNKNOWN, ABI, MDYN, FLUOR, TRUVEL };
 enum Method { MAX_INTEGRAL, MIN_XBANDING };
 enum Status
 {
  STS_UNINITD,
  STS_INITD,
  STS_NO_MEM,
  STS_NO_SIZE,
  STS_NO_PARMF,
  STS_NO_DATAF,
  STS_SCNL_RD_FAIL,
  STS_CORRUPT,
  STS_NO_LNORD,
  STS_IFILE_UN
 };

 Wvfm();
 Wvfm(char const* dfn,char const* pfn,char const* lnordr=NULL,
       DATASRC ds=MDYN, Method m=MAX_INTEGRAL );
 Wvfm( int rows, int cols, char const* lnordr,
    DATASRC ds=MDYN, Method m=MAX_INTEGRAL );
 Wvfm( Wvfm const& rhs );
 Wvfm const& operator=( Wvfm const& rhs );
 ~Wvfm();
 void lnordr( char const* lnordr ):
 void ssm( double const* pssm );
```

```
void ds( DATASRC ds ) { ds_ = ds; }
void smplRate( INTERPOLATE_BY ib ) { ib_ = ib; }
void rows(int r)     { rows_ = r; }
void cols(int c)     { cols_ = c; }
void bgni(int b)     { bgni_ = b; }
void endi(int e)     { endi_ = e; }
void sc_la_set(int s,int l,double v) const {
 if(rows_>cols_) pm_[s][l] = v;
 else        pm_[l][s] = v;
 }
void sc_la_mul(int s.int l,double v) const {
 if(rows_>cols_) pm_[s][l] *= v;
 else        pm_[l][s] *= v;
 }
void append( Wvfm const& rhs, int oscnl, int nscnl );
int       scanl()        const { return rows_>cols_? rows_: cols_; }
int       lanes()        const { return rows_<cols_? rows_: cols_; }
DATASRC   ds()           const { return ds_; }
double    sc_la(int s,int l) const
{ return rows_>cols_? pm_[s][l]: pm_[l][s]; }
double const* ssm()          const { return ssm_[0]; }
void      envelope( ShftVect const& sv ) const;
int       rows()         const { return rows_; }
int       cols()         const { return cols_; }
int       obgni()        const { return obgni_;  }
int       oendi()        const { return oendi_; }
int       bgni()         const { return bgni_; }
int       endi()         const { return endi_; }
double    envv( size_t idx ) const { return (pv_)? pv_[idx]: 0.0; }
int       envi( size_t idx ) const { return (pi_)? pi_[idx]: 0;   }
double    xbnd( size_t idx ) const { return (px_)? px_[idx]: 0.0; }
```

```
float       buzz( size_t idx ) const { return (pb_)? pb_[idx]: 0.0f; }
Method      method()        const { return method_; }
char const* lnordr()        const { return lnordr_; }
void        debug(char const* msg = NULL) const;
int nfeeder( RdrOut& ro, int fVb=0 );
int unstop();
void sort();
Status status()             const { return status_; }
private:
int         rows_;
int         cols_;
int         bgni_;
int         endi_;
DATASRC     ds_;
double      **pm_;

double*     pv_;
int*        pi_;
double*     px_;
float*      pb_;
Method      method_;
char        lnordr_[6];
double      ssm_[4][4];
Status      status_;
INTERPOLATE_BY ib_;
char        fCalcSST_;
int         obgni_,
            oendi_;
int preproc();
int nreader( int Fbw, int finalRead, SegRead& segrd );
void release();
```

```
    void bgnEnd();
    int specSep();
    void sc_la_sub(int s,int l,double v) const {
     if(rows_>cols_) pm_[s][l] -= v;
     else        pm_[l][s] -= v;
     }
    void pm( double **p, int r, int b, int e );
    void transpose();
    void truvelAdjust();
    void leastSquareBaseline_();
    void baseline( int N );
    void noZeros();
    int mdynpre();
    double* dmscanlsum() const;
    double* dmscanlprod() const;
    double* operator[](int i) { return pm_[i]; }
    int t2tBaselineSubt_();
    int  rawPks_( RatioBin& rb) const;
    void bestBaseline_();
    enum DIRECTION_ { BCK=-1, FWD=1 };
    void fbbls_(int fnr, double* b, DIRECTION_ D) const;
};
#endif

/*************************************************************************
 * FILE: xtranorm.cxx -- Extra Normalization to compensate for
 *       a band-lite (as opposed to other band-rich) lanes.
 * AUTHOR:  Andy Marks
* COPYRIGHT (c) 1996, University of Utah
 */
#include <basecall/mb.hxx>
```

```
struct SD
{
  double v;
  int i;
};
double*
Wvfm::dmscanlprod() const
{
  double *p;
  if(NULL != (p = ::dvector(1,scanl())))
   for(int sdx=1; sdx<=scanl(); sdx++) {
    p[sdx] = sc_la(sdx,1);
    for(int ldx=2; ldx<=lanes(); ldx++)
     p[sdx] *= sc_la(sdx,ldx);
    }
  return p;
}
double*
Wvfm::dmscanlsum() const
{
  double *p;
  if(NULL != (p = ::dvector(1,scanl())))
   for(int sdx=1;sdx<=scanl();sdx++) {
    p[sdx] = sc_la( sdx, 1 );
    for(int ldx=2;ldx<=lanes();ldx++)
     p[sdx] += sc_la( sdx, ldx );
    }
  return p;
}
static void
stats( double const* v, int n, double& mn, double& std )
```

```
{
  int idx;
  mn = std = 0.0;
  for(idx=1;idx<=n;idx++)
    mn += v[idx];
  mn /= double(n);
  for(idx=1;idx<=n;idx++) {
    double del = (v[idx]-mn);
    std += del*del;
    }
  std = sqrt( std/double(n) );
}
int
Wvfm::unstop()
{
  int testagain = 1;
  while(1 == testagain) {
    double *p, *s, pmn,pstd, smn,sstd;
    double pmz, smz, geoMean;
    int pmi, smi;
    int ldx, sdx;
    testagain = 0;
    p = dmscanlprod();
    if(NULL == p) return 0;
    s = dmscanlsum();
    if(NULL == s) return 0;
    stats( p, scanl(), pmn, pstd );
    stats( s, scanl(), smn, sstd );
    pmz=p[1]; pmi=1;
    smz=s[1]; smi=1;
    if(0.0 == pstd || 0.0 == sstd) {
```

```
        ::free_dvector( p, 1, scanl() );
        ::free_dvector( s, 1, scanl() );
        return 0;
        }
      for(sdx=1; sdx<=scanl(); sdx++) {
        p[sdx] = (p[sdx]-pmn)/pstd;
        s[sdx] = (s[sdx]-smn)/sstd;
        if(p[sdx] > pmz) { pmz = p[sdx]; pmi = sdx; }
        if(s[sdx] > smz) { smz = s[sdx]; smi = sdx; }
        }
      ::free_dvector( p, 1, scanl() );
      ::free_dvector( s, 1, scanl() );
      if(10 > ::abs(pmi-smi)) {
        geoMean = ::sqrt( pmz * smz );
        if(geoMean > 9.5) {
          int bgn, end;
          testagain = 1;
          bgn = (smi>25)? smi-25: 1;
          end = ((smi+25) < scanl())? smi+25: scanl();
          for(ldx=1; ldx<=4; ldx++)
            for(sdx=bgn; sdx<=end; sdx++)
              sc_la_set( sdx, ldx, 0.0 );
          }
        }
      }
    return 1;
  }
  static int
  sumi2j(int i, int j)
  {
    int t;
```

```
    if(i>j) {
     t = i; i = j; j = t;
     }
    return j*(j+1)/2 - i*(i-1)/2;
}
static int
sd_cmp( void const* e1, void const* e2 )
{
   double d1 = ((SD const*)e1)->v;
   double d2 = ((SD const*)e2)->v;
   if(d1 > d2)     return  1;
   else if(d1 < d2) return -1;
   else            return  0;
}
static int
rankordr( double* pv, int n )
{
   SD* pv2;
   int idx, jdx, hi;
   pv2 = new SD[ n ];
   if(NULL == pv2)
    return 0;
   for(idx=1; idx<=n; idx++) {
    pv2[idx-1].v = pv[idx];
    pv2[idx-1].i = idx;
    }
   ::qsort( pv2, n, sizeof(SD), sd_cmp );
   for(idx=0; idx<n; ) {
    for(hi=idx+1; hi<n; hi++)
     if(pv2[idx].v != pv2[hi].v)
       break;
```

```
      double v = double(sumi2j(idx+1,hi))/double(hi-idx);
      for(jdx=idx;jdx<hi;jdx++)
       pv2[jdx].v = v;
      idx = hi;
      }
    for(idx=0; idx<n; idx++)
     pv[ pv2[idx].i ] = pv2[idx].v;
    delete [] pv2;
    return 1;
}
int
SegRead::xtranorm( Wvfm& bdproc, Wvfm const& raw, int nscanl, int pass2 )
{
   ShftVect& sv = nsv();
   int lane, sdx;
   double gain = 0.0;
   if(!pass2) {
    double bandFreq[5];
    Wvfm rawcopy( raw );
    Wvfm rdycopy( bdproc );
    if(1 != rdycopy.unstop())
     return 0;
    sv = mcalign( rdycopy, sv );
    if(0 == pmb_->remune()) { ShftVect noshft; sv = noshft; }
    pmb_->minFreq_ = 1.0;
    rawcopy.sort();
    bandFreq[0] = pmb_->bandAmpl_[0] = 0.0;
    for(lane=1;lane<=4;lane++)
     pmb_->bandAmpl_[ lane ] = rawcopy.sc_la( 2000, lane );
    rawcopy = raw;
    rawcopy.envelope( sv );
```

```
        for(lane=1;lane<=4;lane++)
        bandFreq[ lane ] = 0.0;
       for(sdx=1+sv.maxshft(); sdx<=nscanl; sdx++)
        bandFreq[ rawcopy.envi( sdx ) ] += 1.0;
      for(lane=1; lane<=4; lane++) {
       bandFreq[ lane ] /= double(nscanl-sv.maxshft()+1);
       if(bandFreq[lane] < pmb_->minFreq_)
        pmb_->minFreq_ = bandFreq[ pmb_->minFreqLane_ = lane ];
       }
      if(1 != ::rankordr( pmb_->bandAmpl_, 4 ))
       return 0;
      }
     if(pmb_->minFreq_ < 0.15) {
      gain = (pmb_->bandAmpl_[pmb_->minFreqLane_] < 2.0)? 0.5: 0.75;
      int NS = bdproc.endi();
      for(sdx=bdproc.bgni();sdx<NS;sdx++)
       bdproc.sc_la_mul( sdx, pmb_->minFreqLane_, gain );
     }
    return 1;
}
```

While the present invention has been described and illustrated in conjunction with a number of specific embodiments, those skilled in the art will appreciate that variations and modifications may be made without departing from the principles of the invention as herein illustrated and described.

The invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. The present embodiments are to be considered in all respects as illustrative, and not restrictive.

V. Claims

1. A method for determining base identity in unprocessed nucleic acid sequencing data, comprising the steps of:

receiving unprocessed input data comprising unprocessed nucleic acid sequencing data;

preprocessing said input data to generate preprocessed data;

blind deconvolving said preprocessed data to generate blind deconvolved data;

extranormalizing said blind deconvolved data to generate extranormalized data;

detecting peaks in said extranormalized data using peak detection means to generate processed data;

editing the quality of said processed data using fuzzy logic editing means to generate called nucleotide sequence and at least one quality value for said called sequence.

2. The method according to claim 1, wherein said preprocessing step further comprises:

identifying Begin and End points in said unprocessed data;

establishing a baseline in said unprocessed data;

subtracting said baseline from said unprocessed data to generate baseline-subtracted data;

separating said baseline-subtracted data to generate preprocessed data, said separating step comprising spectral or leakage separation;

3. The method according to claim 1, wherein said extranormalizing step further comprises:

correcting the relatives mobility of signals in said blind deconvolved data using Monte Carlo alignment means.

4. The method according to claim 3, wherein said extranormalizing step further comprises:

attenuating signals which were accentuated by said blind deconvolving.

5. The method according to claim 1, wherein said peak detection means comprises fuzzy logic insertion detection means to identify and remove putative insertions in said extranormalized data; and

fuzzy logic gapchecking means to identify putative gaps in said extranormalized data and inserting data in said gaps.

6. The method according to claim 5, further comprising:

analysing said extranormalized data with said fuzzy logic insertion detection means before and after analyzing said extranormalized data with said fuzzy logic gap checking means.

7. The method according to claim 1,

wherein said editing means generates at least one quality value by analyzing characteristics of said processed data selected from the group consisting of band height, band width, band shape, band's left gap, band's right gap, cross-banding and baseline buzz.

8. The method according to claim 7,

wherein said editing means generates at least one quality value from said characteristics of said processed data by applying a plurality of fuzzy logic rules.

9. The method of claim 1,

wherein said blind deconvolving is iterative and includes at least a first narrow-band guess for the filter band width value and a refined second band for the filter band width value.

10. A method for identifying DNA sequence in unprocessed nucleic acid sequencing data, comprising the steps of:

receiving unprocessed input data comprising unprocessed nucleic acid sequencing data;

preprocessing said input data to generate preprocessed data;

blind deconvolving said preprocessed data to generate blind deconvolved data;

extranormalizing said blind deconvolved data to generate extranormalized data;

detecting peaks in said extranormalized data to generate peak detected-data;

identifying and removing insertions in said peak detected-data using a fuzzy logic insertion detection algorithm;

identifying and filling gaps in said peak detected-data using a fuzzy logic gap checking algorithm; and

producing processed sequence data.

11. The method of claim 10, further comprising:

editing the quality of said processed sequence data using fuzzy logic editing means to generate called nucleotide sequence and at least one quality value for said called sequence.

12. The method according to claim 10, wherein said preprocessing step further comprises:

identifying Begin and End points in said unprocessed data;

establishing a baseline in said unprocessed data;

subtracting said baseline from said unprocessed data to generate baseline-subtracted data;

separating said baseline-subtracted data to generate preprocessed data, said separating step comprising spectral or leakage separation;

13. The method according to claim 10,

wherein said extranormalizing step further comprises:

correcting the relatives mobility of signals in said blind deconvolved data using a Monte Carlo alignment.

14. The method according to claim 13, wherein said extranormalizing step further comprises:

attenuating signals accentuated by blind deconvolving.

15. The method according to claim 10, further comprising:

analyzing said extranormalized data with said fuzzy logic insertion detection algorithm before and after analyzing said extranormalized data with said fuzzy logic gap checking algorithm.

16. The method according to claim 11, wherein said editing means generates at least one quality value by analyzing characteristics of said processed data selected from the group consisting of band height, band width, band shape, band's left gap, band's right gap, cross-banding and baseline buzz.

17. The method according to claim 16, wherein said editing means generates at least one quality value from said characteristics of said processed data by applying a plurality of fuzzy logic rules.

18. The method of claim 10,

wherein said blind deconvolving is iterative and includes at least a first narrow-band guess for the filter band width value and a refined, second guess for the filter band width value.

19. A method of determining a nucleotide sequence of a DNA molecule comprising:

providing a set of lane signals encoding the migration pattern of a DNA molecule subjected to DNA sequence analysis to generate an input data;

preprocessing said input data to generate preprocessed data, said preprocessing comprising at least one of the following steps:

identifying Begin and End points,

subtracting baseline noise,

spectrally separating said input data using a separation matrix to correct for spectral cross-talk, and

leakage separating said input data using a separation matrix to correct for lane leakage;

blind deconvolving said preprocessed data to generate blind deconvolved data, said blind deconvolving deblurring signals in said preprocessed data and normalizing signal amplitudes, said blind deconvolving using an iterative filter band width algorithm;

extranormalizing said blind deconvolved data to generate extranormalized data, said extranormalizing including at least one of the following steps:

correcting relative signal mobility differences using a Monte Carlo alignment, and

attenuating signals accentuated by blind deconvolution;

detecting peaks in said extranormalized data to generate peak detected-data;

identifying and removing insertions in said peak-detected data using fuzzy logic insertion detection algorithm;

identifying and filling gaps in said peak-detected data using fuzzy logic gap checking algorithm;

producing processed sequence data.

20. A digital computer system programmed to perform method for identifying DNA sequence in unprocessed nucleic acid sequencing data, said digital computer system including:

a central processing unit,

dynamic memory, and

means for outputting data,

the method comprising:

receiving unprocessed input data comprising unprocessed nucleic acid sequencing data;

preprocessing said input data to generate preprocessed data;

blind deconvolving said preprocessed data to generate blind deconvolved data;

extranormalizing said blind deconvolved data to generate extranormalized data;

detecting peaks in said extranormalized data to generate peak detected-data;

identifying and removing insertions in said peak detected-data using a fuzzy logic insertion detection algorithm;

identifying and filling gaps in said peak detected-data using a fuzzy logic gap checking algorithm;

producing processed sequence data, and

editing the quality of said processed data using fuzzy logic editing means to generate called nucleotide sequence and at least one quality value for said called sequence.

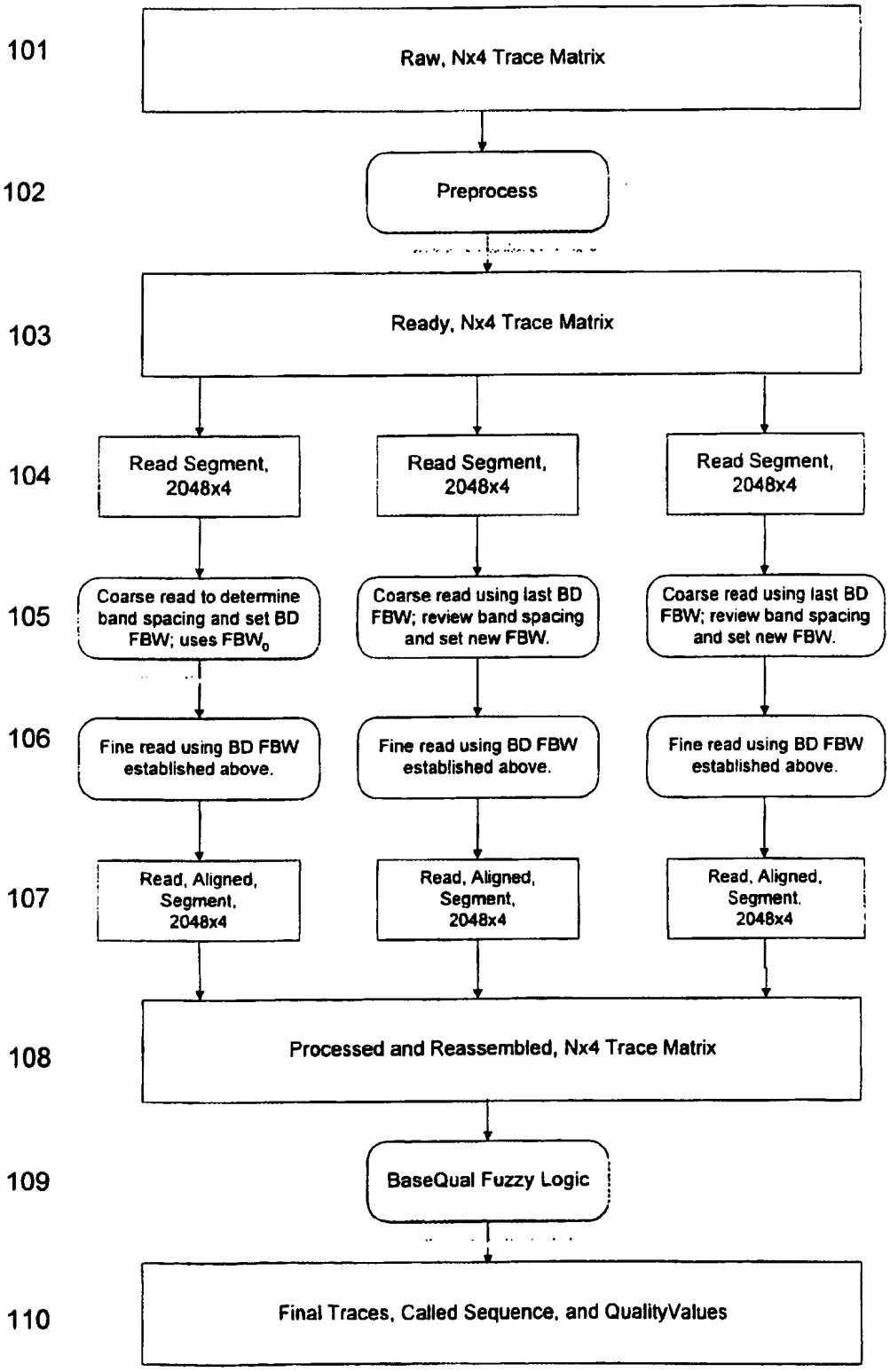
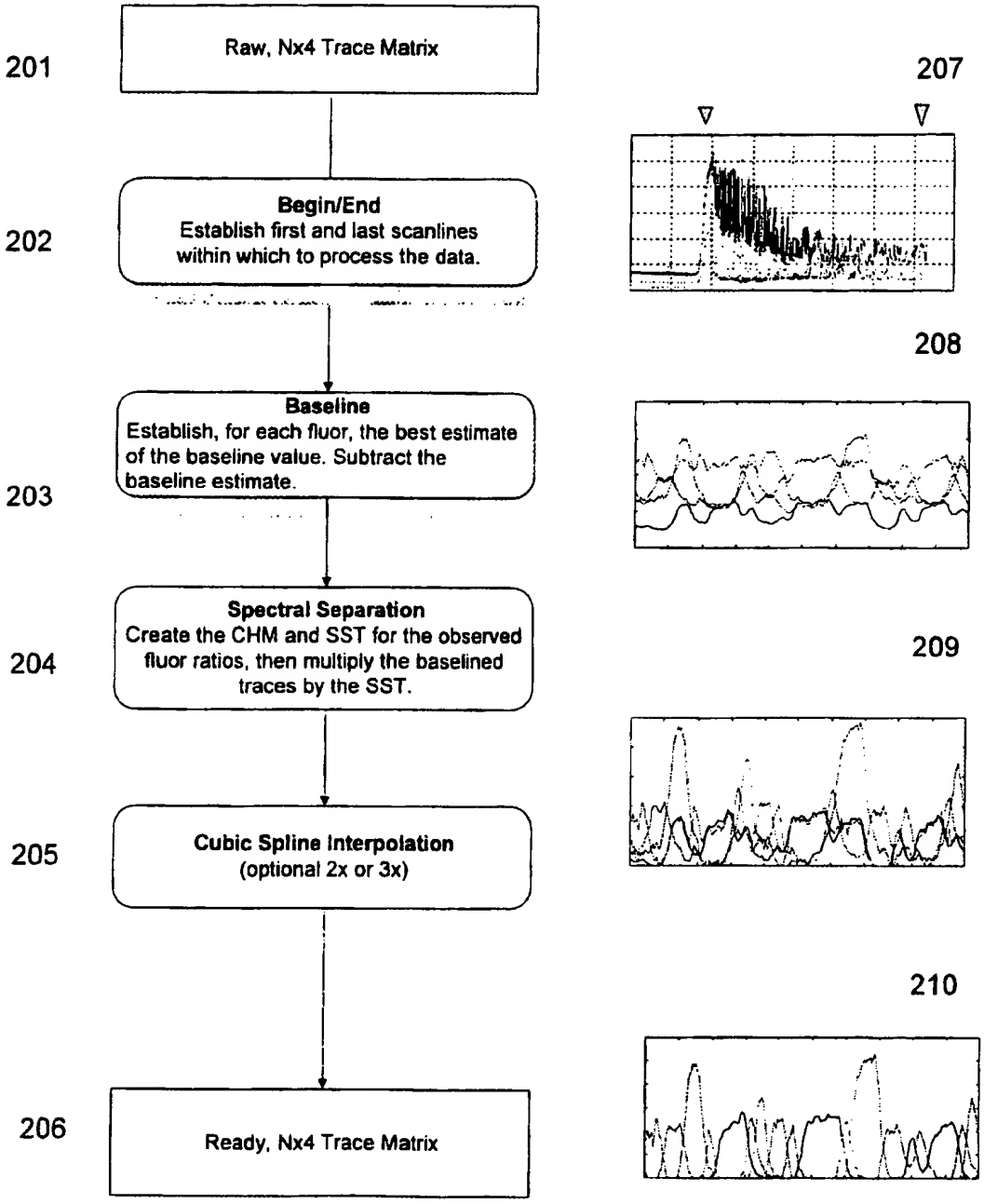
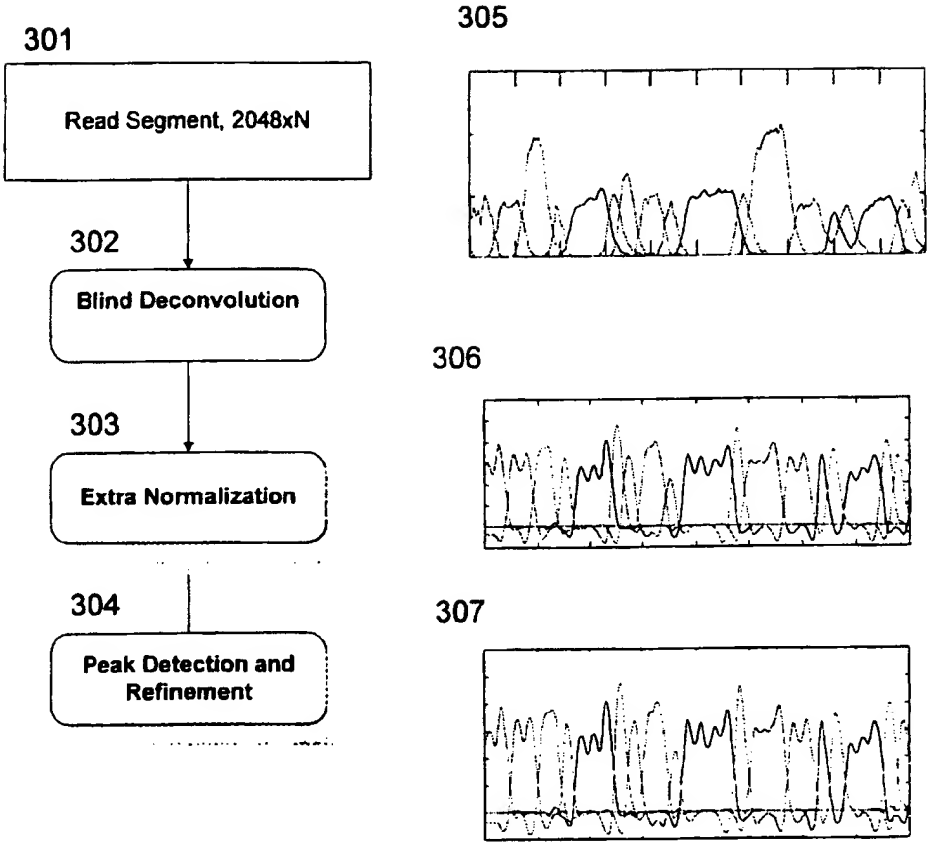

101
Raw, Nx4 Trace Matrix
102
Preprocess
103
Ready, Nx4 Trace Matrix
104
Read Segment, 2048x4
Read Segment, 2048x4
Read Segment, 2048x4
105
Coarse read to determine band spacing and set BD FBW; uses FBW0
Coarse read using last BD FBW; review band spacing and set new FBW.
Coarse read using last BD FBW; review band spacing and set new FBW.
106
Fine read using BD FBW established above.
Fine read using BD FBW established above.
Fine read using BD FBW established above.
107
Read, Aligned, Segment, 2048x4
Read, Aligned, Segment, 2048x4
Read, Aligned, Segment, 2048x4
108
Processed and Reassembled, Nx4 Trace Matrix
109
BaseQual Fuzzy Logic
110
Final Traces, Called Sequence, and QualityValues

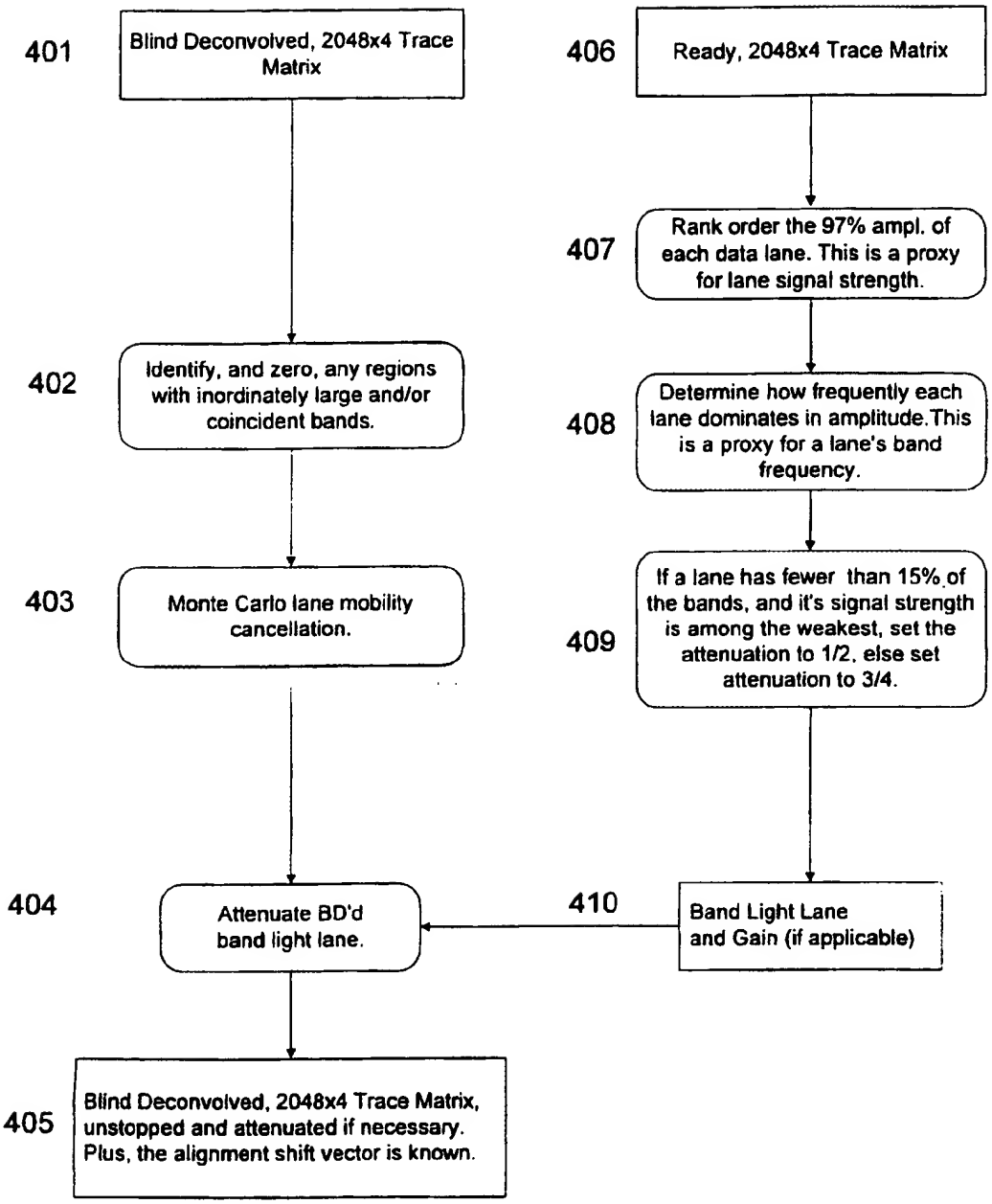
Figure 1


201
Raw, Nx4 Trace Matrix
202
Begin/End
Establish first and last scanlines within which to process the data.
203
Baseline
Establish, for each fluor, the best estimate of the baseline value. Subtract the baseline estimate.
204
Spectral Separation
Create the CHM and SST for the observed fluor ratios, then multiply the baselined traces by the SST.
205
Cubic Spline Interpolation
(optional 2x or 3x)
206
Ready, Nx4 Trace Matrix
207
208
209
210

Figur 2


301
Read Segment, 2048xN
302
Blind Deconvolution
303
Extra Normalization
304
Peak Detection and Refinement
305
306
307

Figur 3


401
Blind Deconvolved, 2048x4 Trace Matrix
402
Identify, and zero, any regions with inordinately large and/or coincident bands.
403
Monte Carlo lane mobility cancellation.
404
Attenuate BD'd band light lane.
405
Blind Deconvolved, 2048x4 Trace Matrix, unstopped and attenuated if necessary. Plus, the alignment shift vector is known.
406
Ready, 2048x4 Trace Matrix
407
Rank order the 97% ampl. of each data lane. This is a proxy for lane signal strength.
408
Determine how frequently each lane dominates in amplitude.This is a proxy for a lane's band frequency.
409
If a lane has fewer than 15% of the bands, and it's signal strength is among the weakest, set the attenuation to 1/2, else set attenuation to 3/4.
410
Band Light Lane and Gain (if applicable)

Figur 4

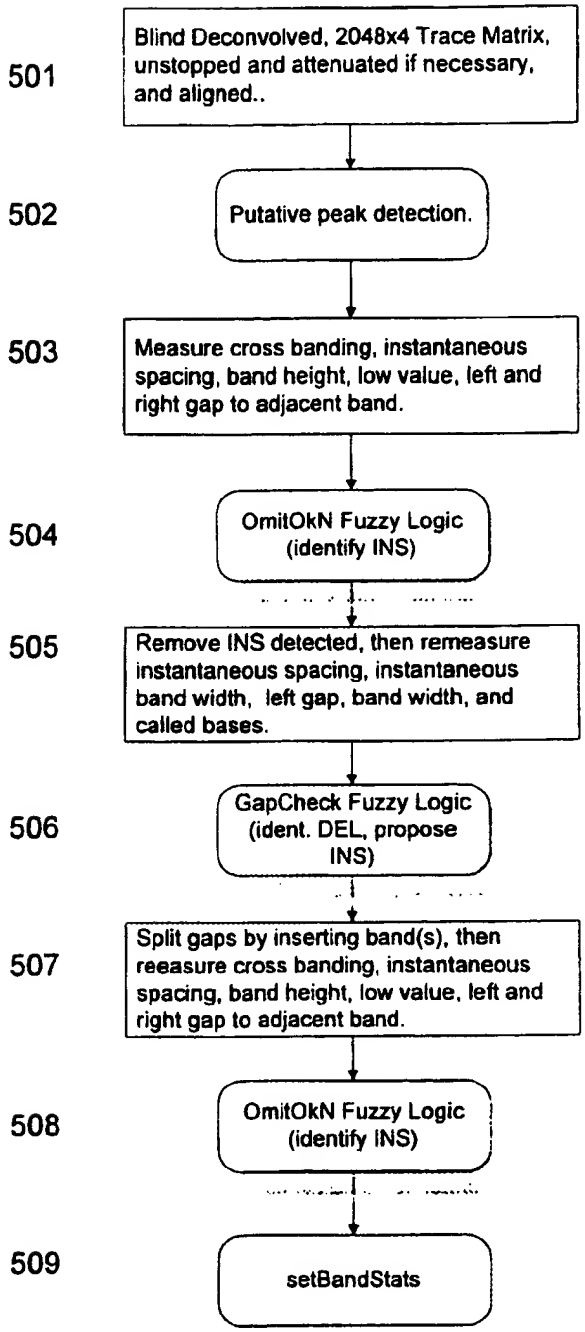

501
Blind Deconvolved, 2048x4 Trace Matrix, unstopped and attenuated if necessary, and aligned..
502
Putative peak detection.
503
Measure cross banding, instantaneous spacing, band height, low value, left and right gap to adjacent band.
504
OmitOkN Fuzzy Logic (identify INS)
505
Remove INS detected, then remeasure instantaneous spacing, instantaneous band width, left gap, band width, and called bases.
506
GapCheck Fuzzy Logic (ident. DEL, propose INS)
507
Split gaps by inserting band(s), then reeasure cross banding, instantaneous spacing, band height, low value, left and right gap to adjacent band.
508
OmitOkN Fuzzy Logic (identify INS)
509
setBandStats

Figure 5

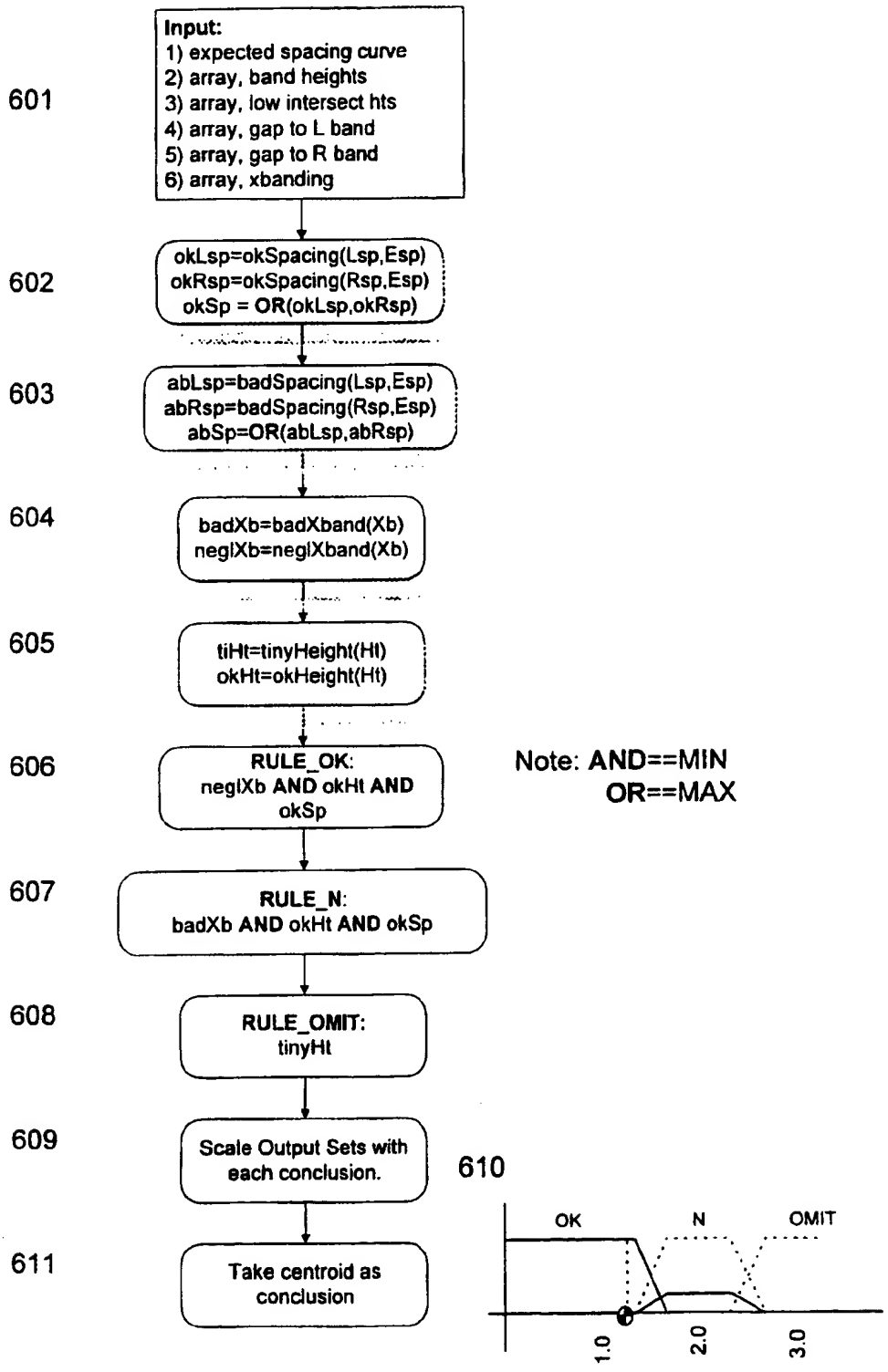

Input:
1) expected spacing curve
2) array, band heights
3) array, low intersect hts
4) array, gap to L band
5) array, gap to R band
6) array, xbanding
601
okLsp=okSpacing(Lsp,Esp)
okRsp=okSpacing(Rsp,Esp)
okSp = OR(okLsp,okRsp)
602
abLsp=badSpacing(Lsp,Esp)
abRsp=badSpacing(Rsp,Esp)
abSp=OR(abLsp,abRsp)
603
badXb=badXband(Xb)
negIXb=negIXband(Xb)
604
tiHt=tinyHeight(Ht)
okHt=okHeight(Ht)
605
RULE_OK:
negIXb AND okHt AND okSp
606
Note: AND==MIN
OR==MAX
RULE_N:
badXb AND okHt AND okSp
607
RULE_OMIT:
tinyHt
608
Scale Output Sets with each conclusion.
609
610
OK
N
OMIT
1.0
2.0
3.0
Take centroid as conclusion
611

Figure 6

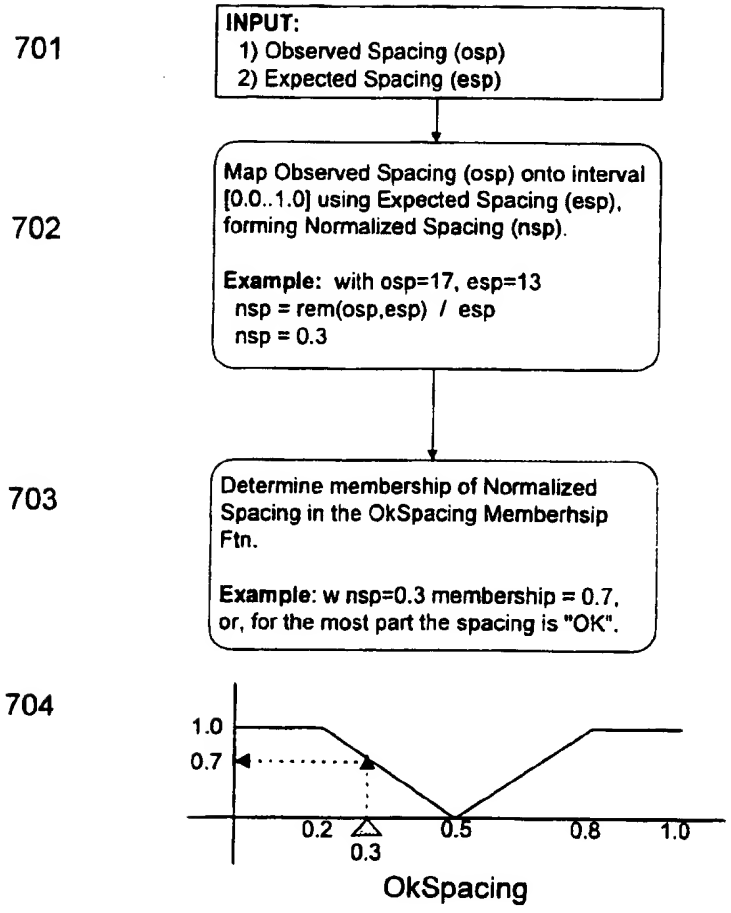

701
INPUT:
1) Observed Spacing (osp)
2) Expected Spacing (esp)
702
Map Observed Spacing (osp) onto interval [0.0..1.0] using Expected Spacing (esp), forming Normalized Spacing (nsp).
Example: with osp=17, esp=13
nsp = rem(osp,esp) / esp
nsp = 0.3
703
Determine membership of Normalized Spacing in the OkSpacing Memberhsip Ftn.
Example: w nsp=0.3 membership = 0.7, or, for the most part the spacing is "OK".
704
1.0
0.7
0.2
0.3
0.5
0.8
1.0
OkSpacing

Figure 7

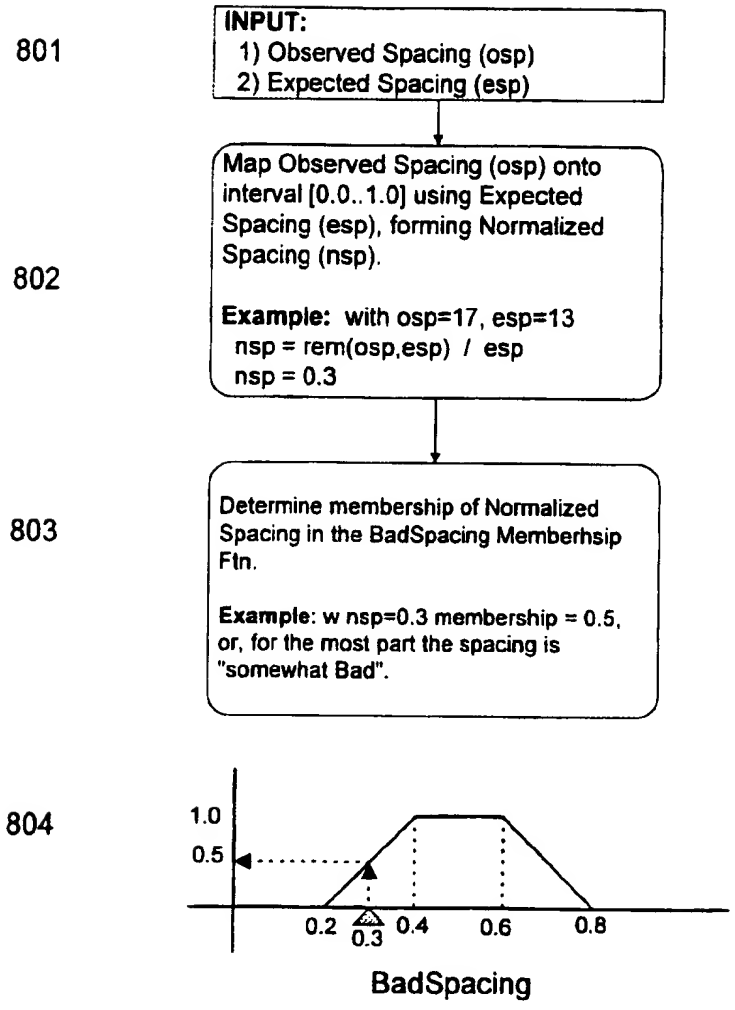
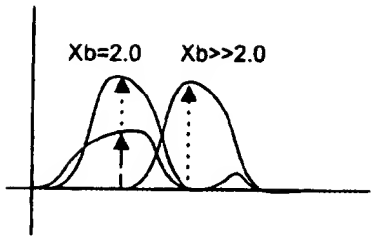

801
INPUT:
1) Observed Spacing (osp)
2) Expected Spacing (esp)
802
Map Observed Spacing (osp) onto interval [0.0..1.0] using Expected Spacing (esp), forming Normalized Spacing (nsp).
Example: with osp=17, esp=13
nsp = rem(osp,esp) / esp
nsp = 0.3
803
Determine membership of Normalized Spacing in the BadSpacing Memberhsip Ftn.
Example: w nsp=0.3 membership = 0.5, or, for the most part the spacing is "somewhat Bad".
804
1.0
0.5
0.2
0.3
0.4
0.6
0.8
BadSpacing

Figure 8

901



902

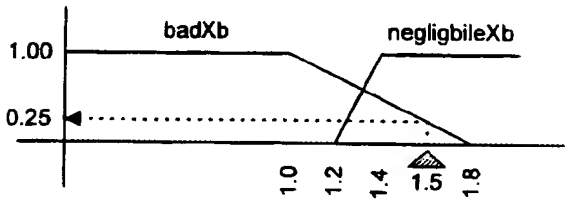
INPUT:
1) Crossbanding measurement, or the band amplitude divided by the amplitude of the next most prominent trace.

903

Determine membership of Absolute Cross Banding (Axb) in the CrossBanding Membership Ftns.

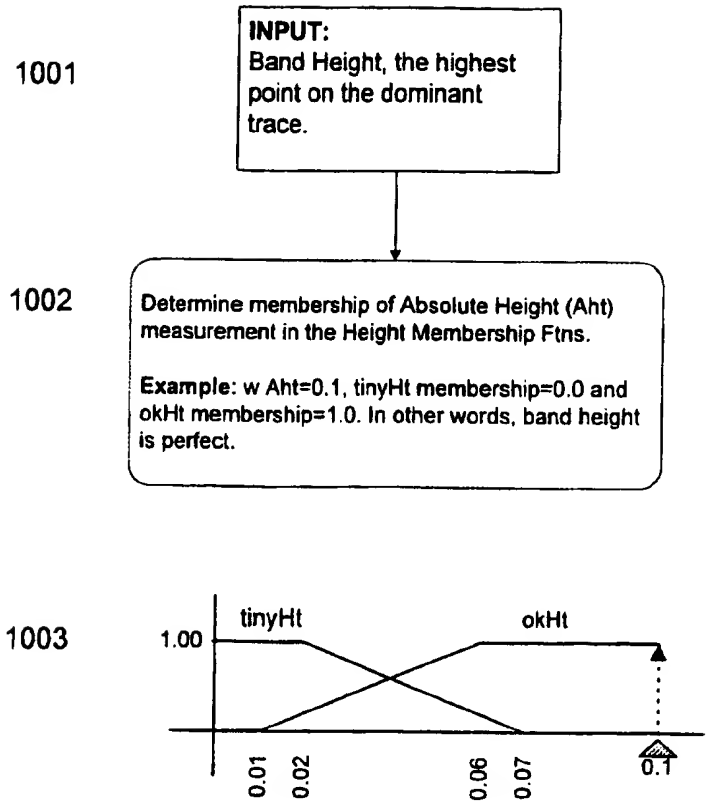
Example: w Axb=1.5, badXb membership=0.25 and negligibleXb membership=1.0. In other words, crossbanding is ok, but is not perfect.

904



Cross Banding

Figur 9


1001
INPUT:
Band Height, the highest point on the dominant trace.
1002
Determine membership of Absolute Height (Aht) measurement in the Height Membership Ftns.
Example: w Aht=0.1, tinyHt membership=0.0 and okHt membership=1.0. In other words, band height is perfect.
1003
tinyHt
okHt
1.00
0.01
0.02
0.06
0.07
0.1

Figur 10

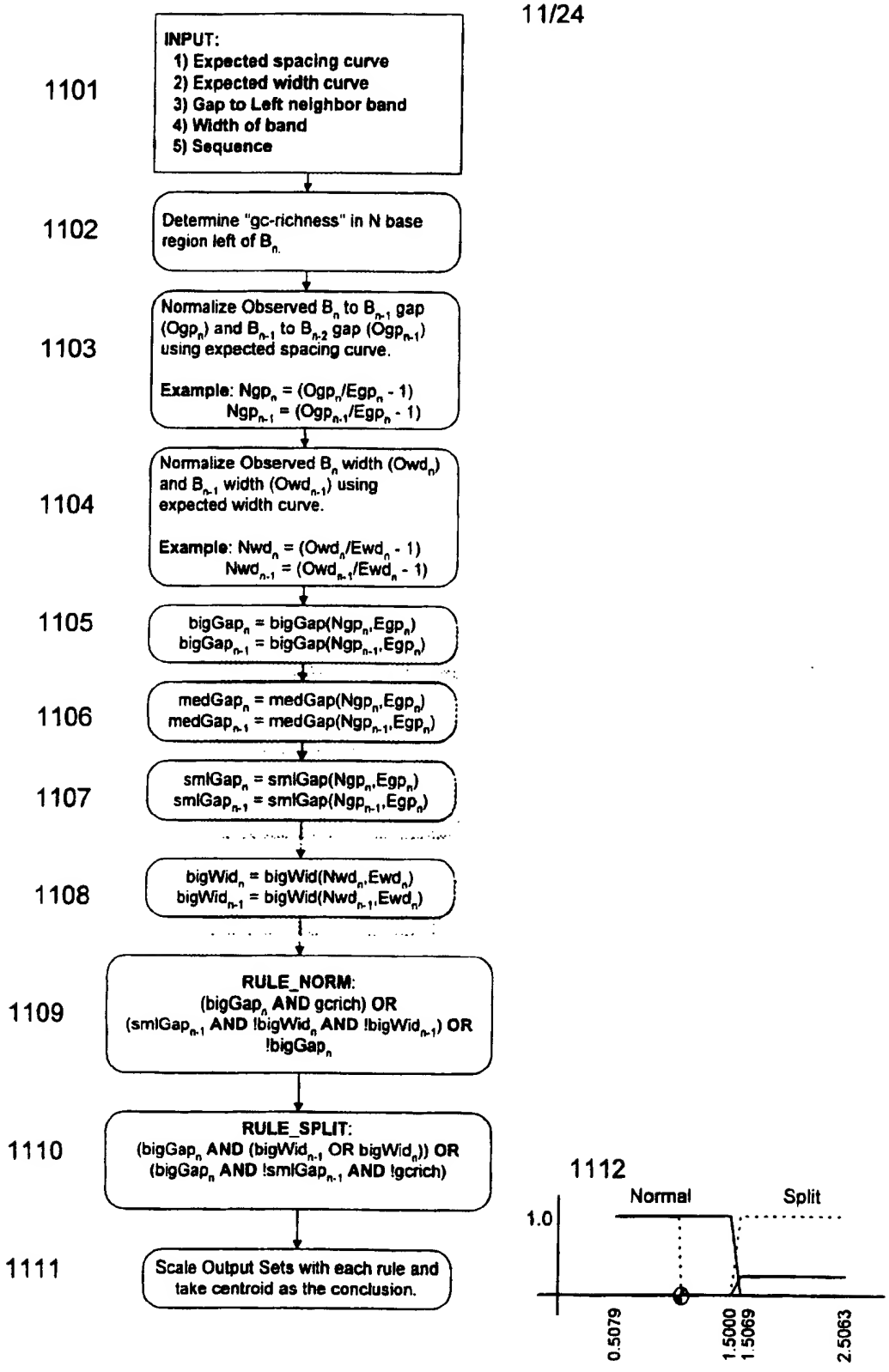
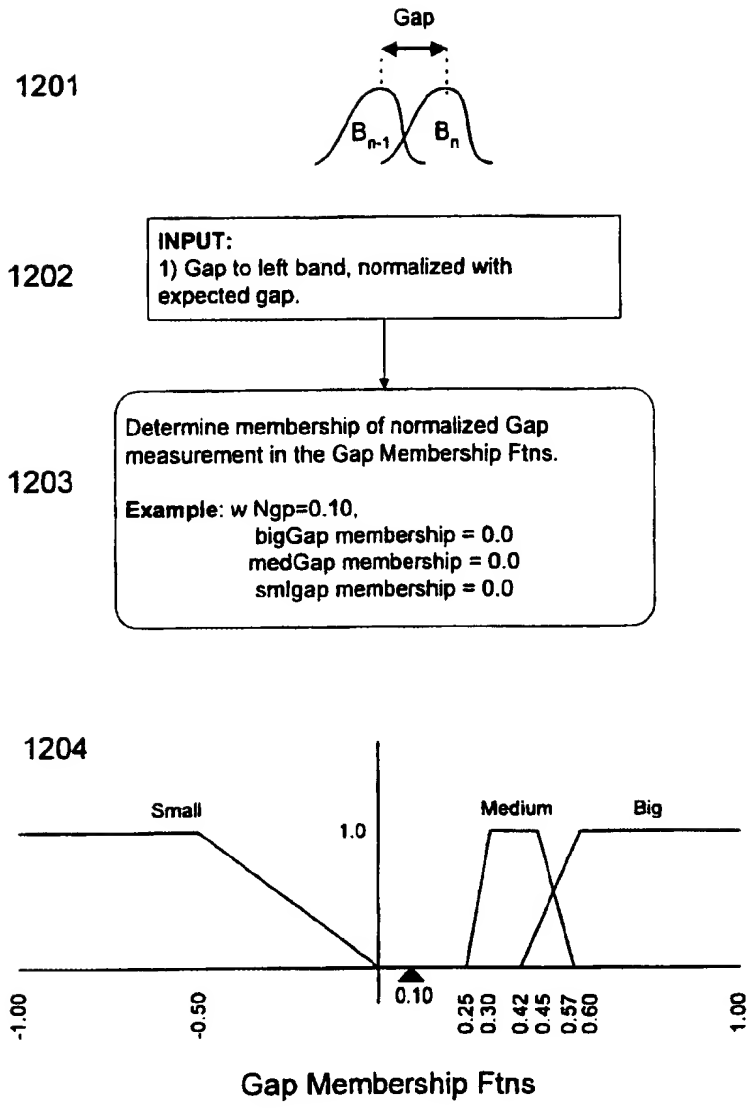

1101
INPUT:
1) Expected spacing curve
2) Expected width curve
3) Gap to Left neighbor band
4) Width of band
5) Sequence
1102
Determine "gc-richness" in N base region left of B_n.
1103
Normalize Observed B_n to B_{n-1} gap (Ogp_n) and B_{n-1} to B_{n-2} gap (Ogp_{n-1}) using expected spacing curve.
Example: $Ngp_n = (Ogp_n/Egp_n - 1)$
$Ngp_{n-1} = (Ogp_{n-1}/Egp_n - 1)$
1104
Normalize Observed B_n width (Owd_n) and B_{n-1} width (Owd_{n-1}) using expected width curve.
Example: $Nwd_n = (Owd_n/Ewd_n - 1)$
$Nwd_{n-1} = (Owd_{n-1}/Ewd_n - 1)$
1105
$bigGap_n = bigGap(Ngp_n, Egp_n)$
$bigGap_{n-1} = bigGap(Ngp_{n-1}, Egp_n)$
1106
$medGap_n = medGap(Ngp_n, Egp_n)$
$medGap_{n-1} = medGap(Ngp_{n-1}, Egp_n)$
1107
$smlGap_n = smlGap(Ngp_n, Egp_n)$
$smlGap_{n-1} = smlGap(Ngp_{n-1}, Egp_n)$
1108
$bigWid_n = bigWid(Nwd_n, Ewd_n)$
$bigWid_{n-1} = bigWid(Nwd_{n-1}, Ewd_n)$
1109
RULE_NORM:
($bigGap_n$ AND gcrich) OR
($smlGap_{n-1}$ AND !$bigWid_n$ AND !$bigWid_{n-1}$) OR
!$bigGap_n$
1110
RULE_SPLIT:
($bigGap_n$ AND ($bigWid_{n-1}$ OR $bigWid_n$)) OR
($bigGap_n$ AND !$smlGap_{n-1}$ AND !gcrich)
1111
Scale Output Sets with each rule and take centroid as the conclusion.
1112
Normal
Split
1.0
0.5079
1.5000
1.5069
2.5063

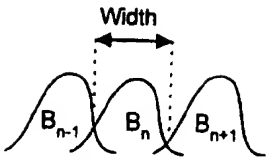
Figure 11



Gap Membership Ftns

Figure 12

1301



1302

INPUT:
1) Width of a band, normalized with Expected Width

1303

Determine membership of normalized Wdith measurement in the Width Membership Ftn.

Example: w Nwd=0.20, bigWid membership = 0.20

1304

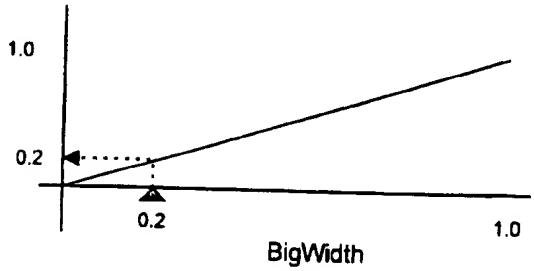
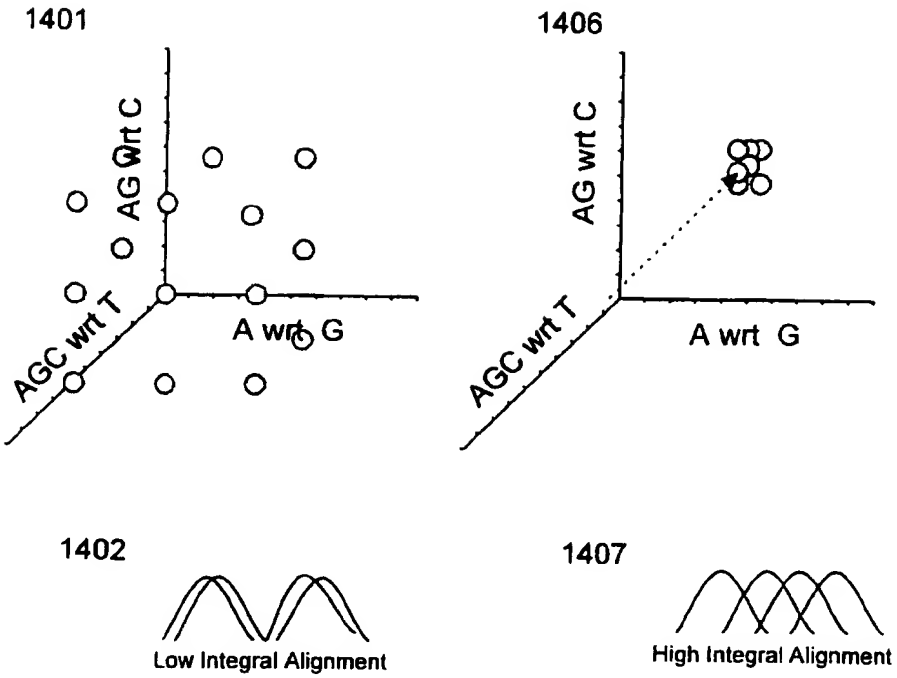


Figure 13


1401
AG wrt C
AGC wrt T
A wrt G
1406
AG wrt C
AGC wrt T
A wrt G
1402
Low Integral Alignment
1407
High Integral Alignment

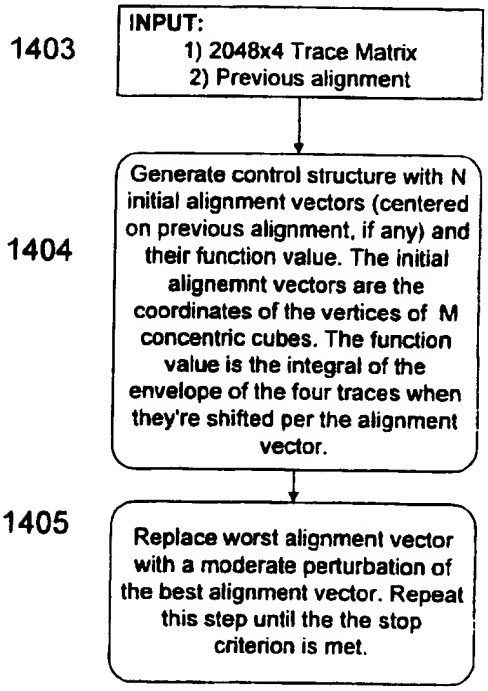
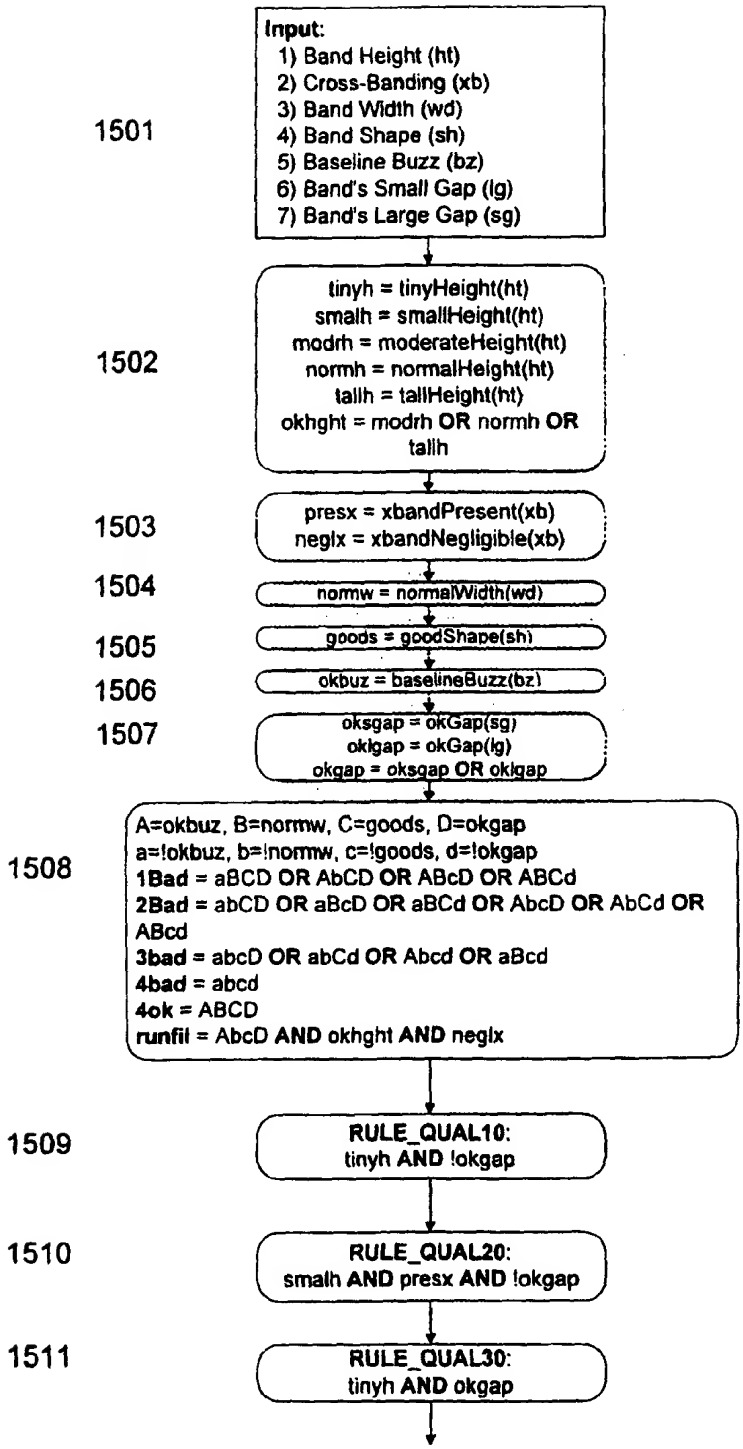

1403
INPUT:
1) 2048x4 Trace Matrix
2) Previous alignment
1404
Generate control structure with N initial alignment vectors (centered on previous alignment, if any) and their function value. The initial alignemnt vectors are the coordinates of the vertices of M concentric cubes. The function value is the integral of the envelope of the four traces when they're shifted per the alignment vector.
1405
Replace worst alignment vector with a moderate perturbation of the best alignment vector. Repeat this step until the the stop criterion is met.

Figure 14


Input:
1) Band Height (ht)
2) Cross-Banding (xb)
3) Band Width (wd)
4) Band Shape (sh)
5) Baseline Buzz (bz)
6) Band's Small Gap (lg)
7) Band's Large Gap (sg)
1501
tinyh = tinyHeight(ht)
smalh = smallHeight(ht)
modrh = moderateHeight(ht)
normh = normalHeight(ht)
tallh = tallHeight(ht)
okhght = modrh OR normh OR tallh
1502
presx = xbandPresent(xb)
negIx = xbandNegligible(xb)
1503
normw = normalWidth(wd)
1504
goods = goodShape(sh)
1505
okbuz = baselineBuzz(bz)
1506
oksgap = okGap(sg)
oklgap = okGap(lg)
okgap = oksgap OR oklgap
1507
A=okbuz, B=normw, C=goods, D=okgap
a=!okbuz, b=!normw, c=!goods, d=!okgap
1Bad = aBCD OR AbCD OR ABcD OR ABCd
2Bad = abCD OR aBcD OR aBCd OR AbcD OR AbCd OR ABcd
3bad = abcD OR abCd OR Abcd OR aBcd
4bad = abcd
4ok = ABCD
runfil = AbcD AND okhght AND negIx
1508
RULE_QUAL10:
tinyh AND !okgap
1509
RULE_QUAL20:
smalh AND presx AND !okgap
1510
RULE_QUAL30:
tinyh AND okgap
1511

Figur 15

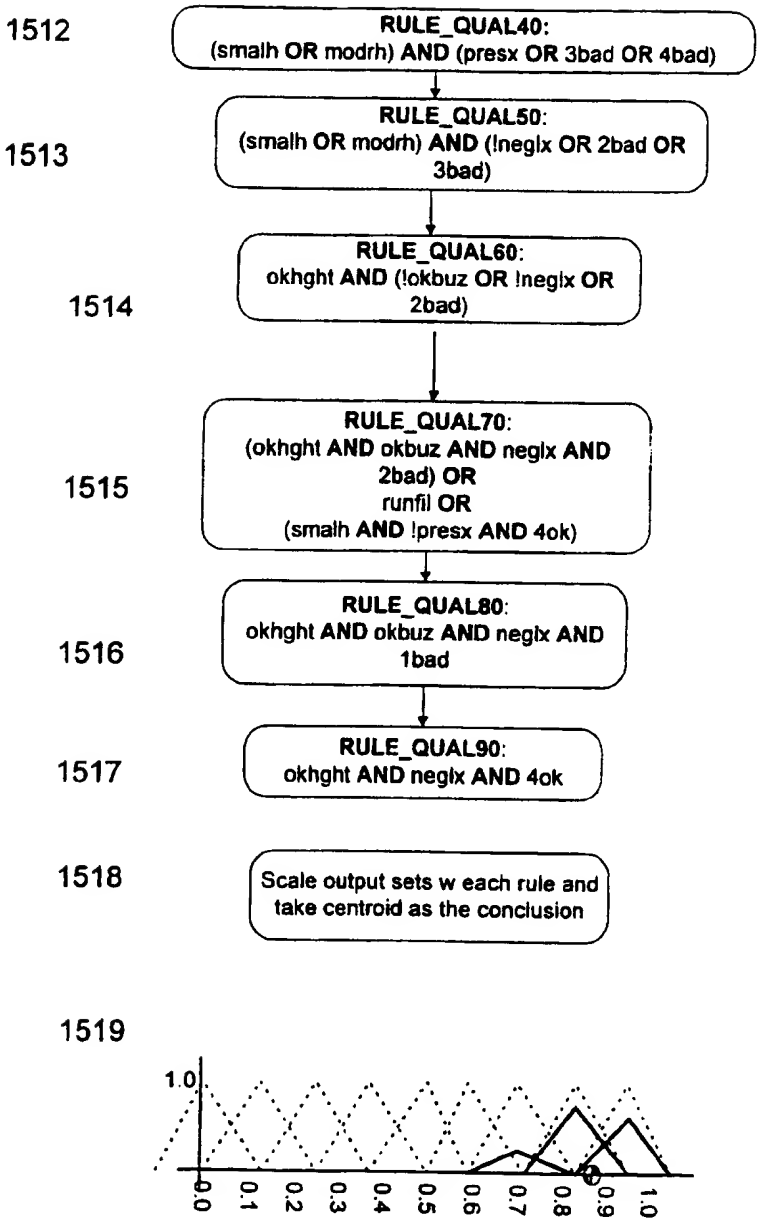

1512
RULE_QUAL40:
(smalh OR modrh) AND (presx OR 3bad OR 4bad)
1513
RULE_QUAL50:
(smalh OR modrh) AND (!negIx OR 2bad OR 3bad)
1514
RULE_QUAL60:
okhght AND (!okbuz OR !negIx OR 2bad)
1515
RULE_QUAL70:
(okhght AND okbuz AND negIx AND 2bad) OR
runfil OR
(smalh AND !presx AND 4ok)
1516
RULE_QUAL80:
okhght AND okbuz AND negIx AND 1bad
1517
RULE_QUAL90:
okhght AND negIx AND 4ok
1518
Scale output sets w each rule and take centroid as the conclusion
1519
1.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Figure 15

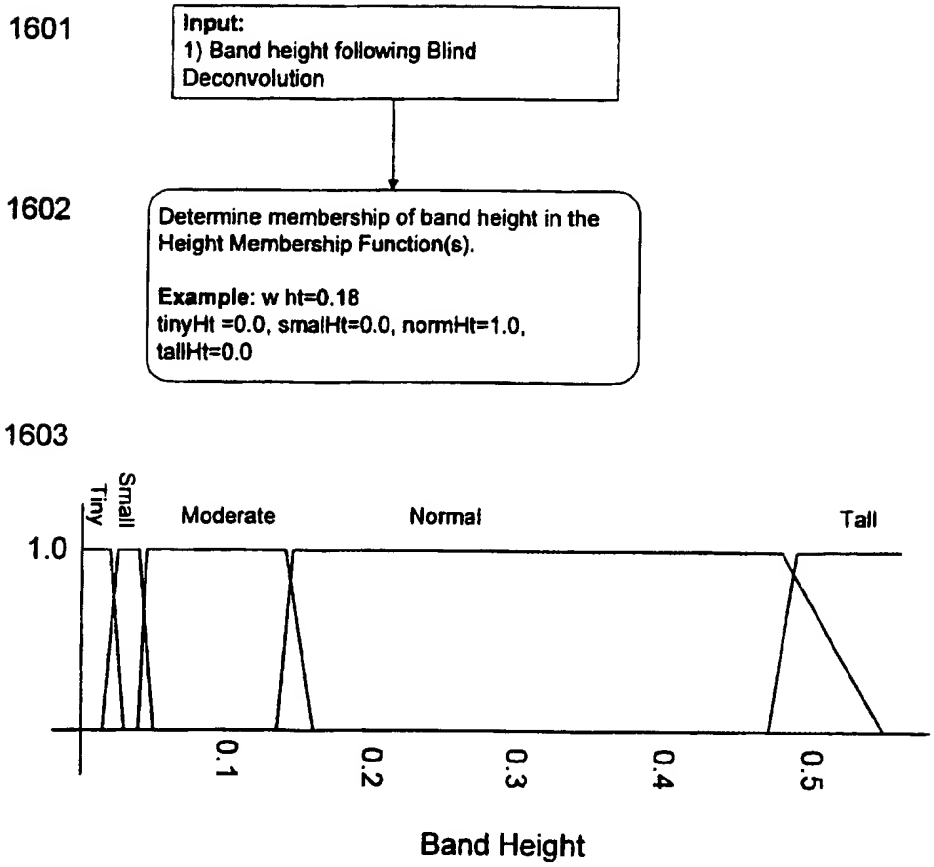
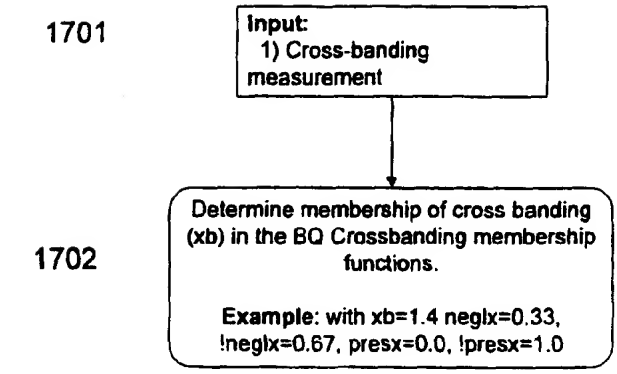

1601
Input:
1) Band height following Blind Deconvolution
1602
Determine membership of band height in the Height Membership Function(s).
Example: w ht=0.18
tinyHt =0.0, smallHt=0.0, normHt=1.0, tallHt=0.0
1603
Tiny
Small
Moderate
Normal
Tall
1.0
0.1
0.2
0.3
0.4
0.5
Band Height

Figure 16


1701
Input:
1) Cross-banding measurement
1702
Determine membership of cross banding (xb) in the BQ Crossbanding membership functions.
Example: with xb=1.4 negIx=0.33, !negIx=0.67, presx=0.0, !presx=1.0

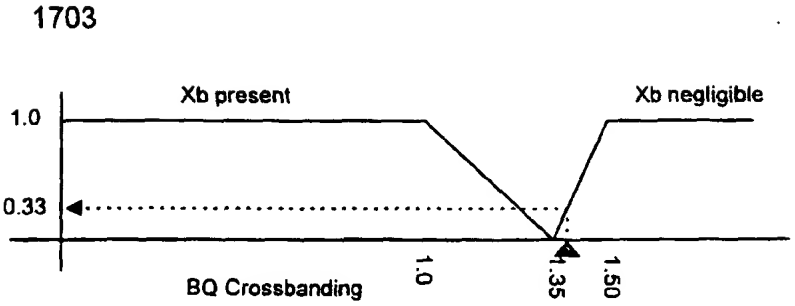

1703
Xb present
Xb negligible
1.0
0.33
BQ Crossbanding
1.0
1.35
1.50

Figure 17

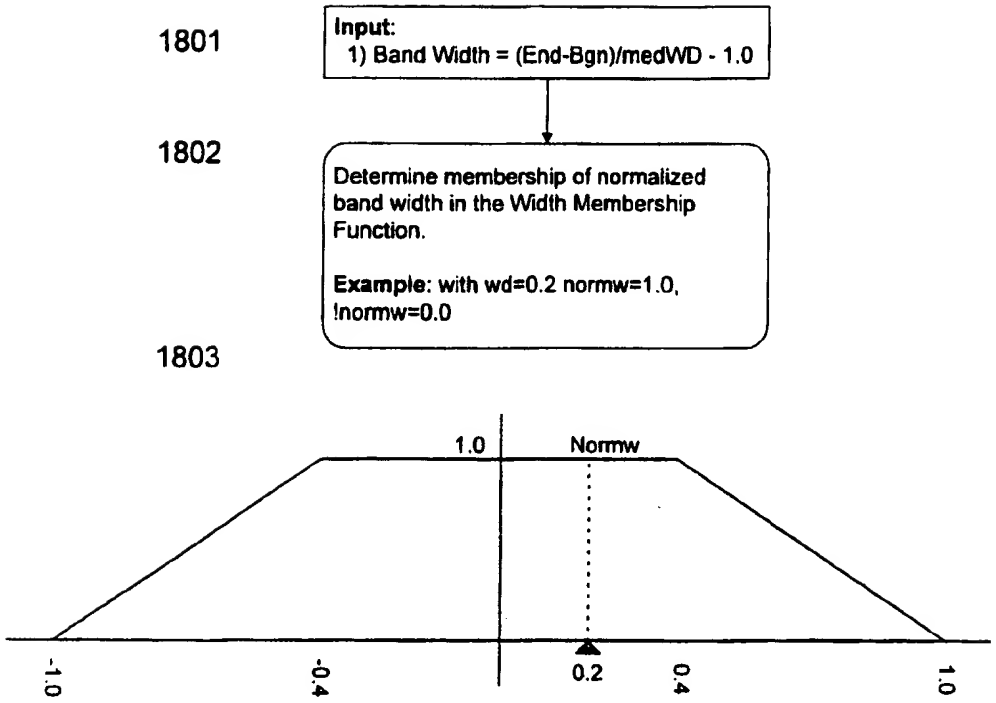

1801
Input:
1) Band Width = (End-Bgn)/medWD - 1.0
1802
Determine membership of normalized band width in the Width Membership Function.
Example: with wd=0.2 normw=1.0, !normw=0.0
1803
1.0
Normw
-1.0
-0.4
0.2
0.4
1.0

Figure 18

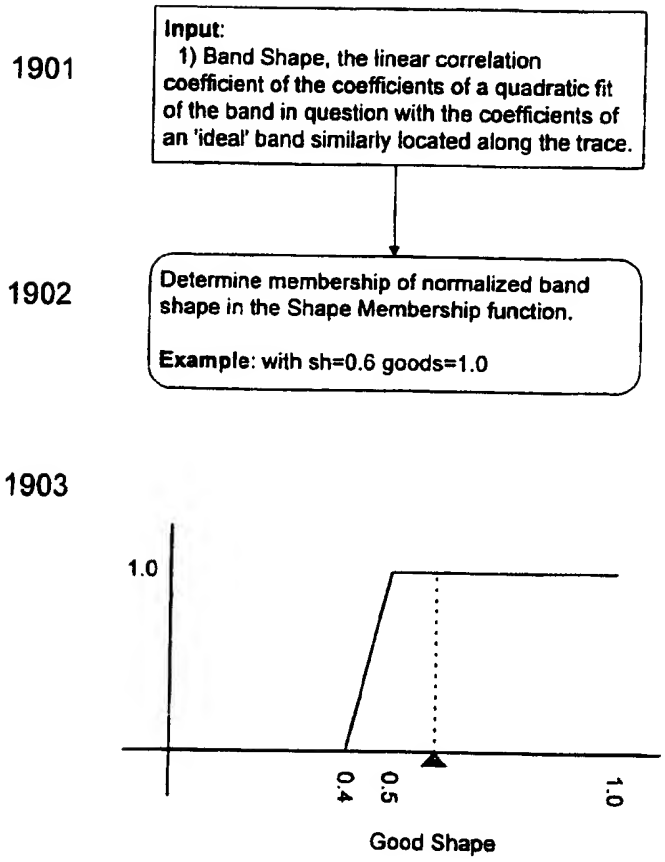

1901
Input:
1) Band Shape, the linear correlation coefficient of the coefficients of a quadratic fit of the band in question with the coefficients of an 'ideal' band similarly located along the trace.
1902
Determine membership of normalized band shape in the Shape Membership function.
Example: with sh=0.6 goods=1.0
1903
1.0
0.4
0.5
1.0
Good Shape

Figure 19

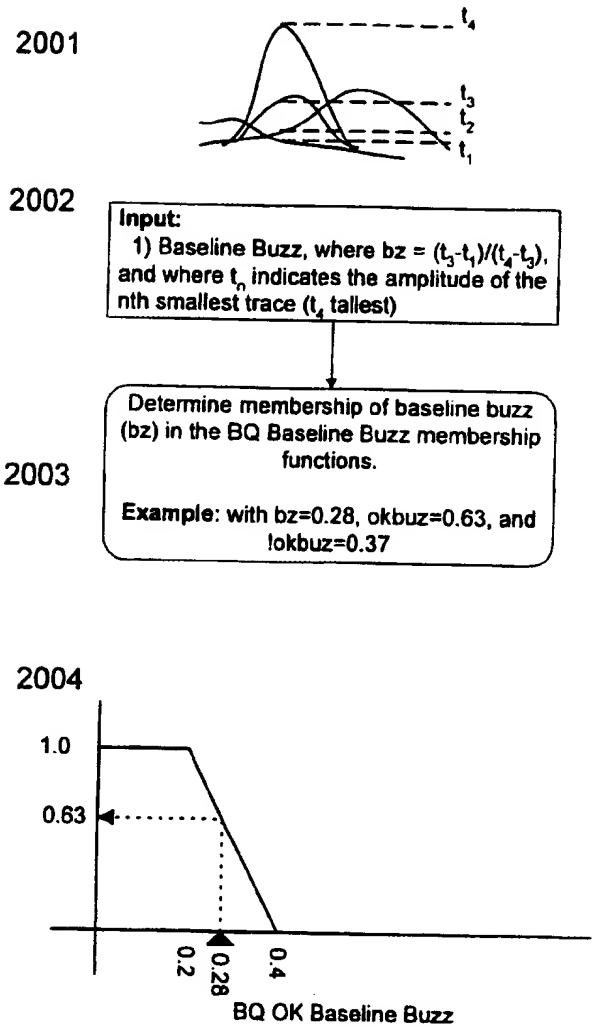

2001
t_4
t_3
t_2
t_1
2002
Input:
1) Baseline Buzz, where bz = $(t_3-t_1)/(t_4-t_3)$, and where t_n indicates the amplitude of the nth smallest trace (t_4 tallest)
2003
Determine membership of baseline buzz (bz) in the BQ Baseline Buzz membership functions.
Example: with bz=0.28, okbuz=0.63, and !okbuz=0.37
2004
1.0
0.63
0.2
0.28
0.4
BQ OK Baseline Buzz

Figure 20

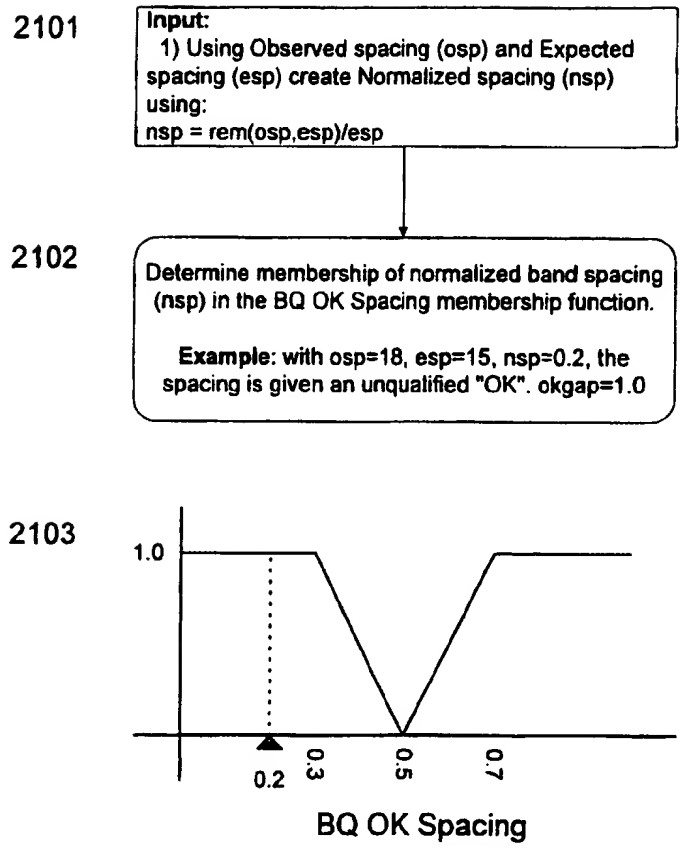
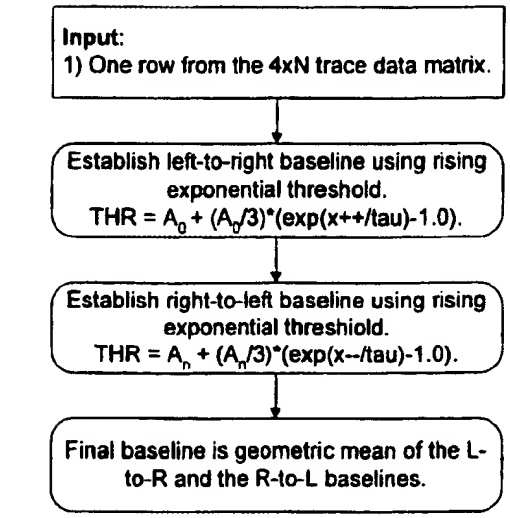
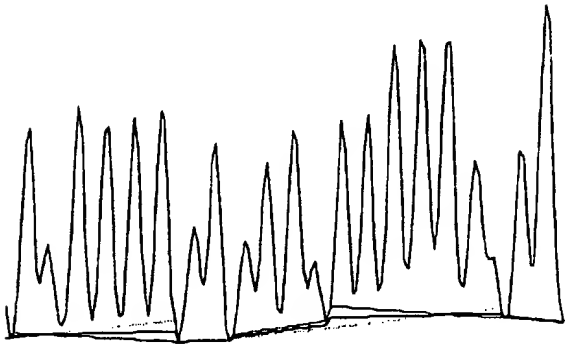
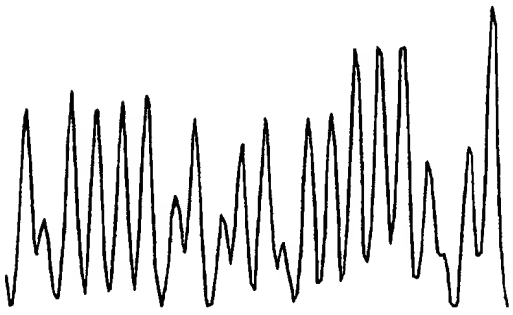

2101
Input:
1) Using Observed spacing (osp) and Expected spacing (esp) create Normalized spacing (nsp) using:
nsp = rem(osp,esp)/esp
2102
Determine membership of normalized band spacing (nsp) in the BQ OK Spacing membership function.
Example: with osp=18, esp=15, nsp=0.2, the spacing is given an unqualified "OK". okgap=1.0
2103
1.0
0.2
0.3
0.5
0.7
BQ OK Spacing

Figure 21





2206 Forward, Backward, and Combined.



Baselin subtracted

Figur 22

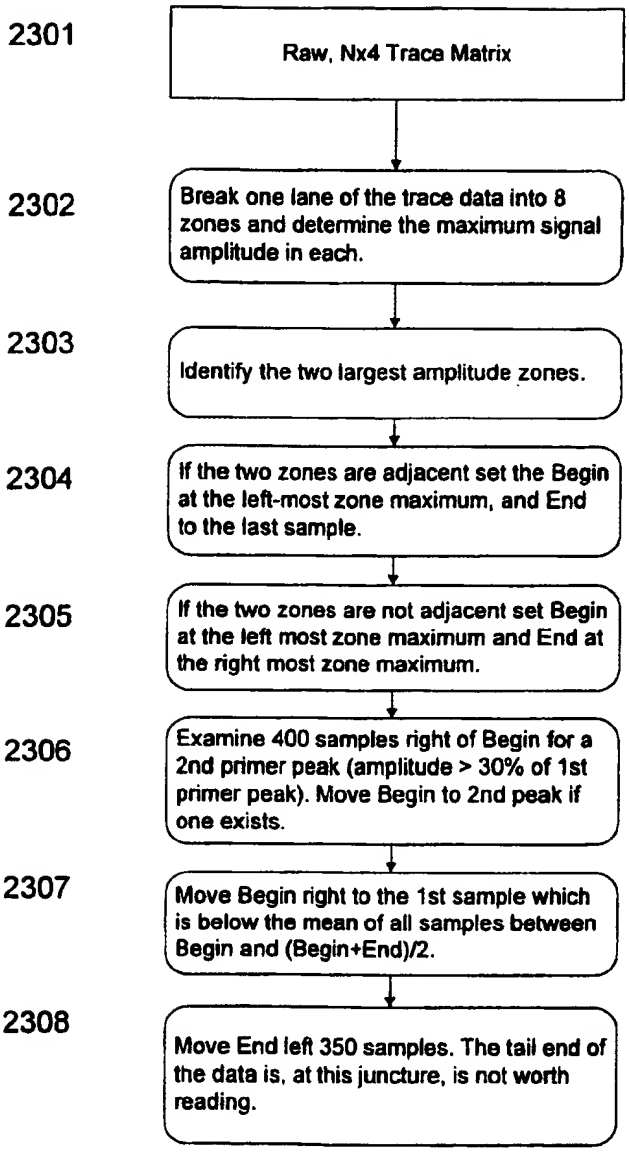

2301
Raw, Nx4 Trace Matrix
2302
Break one lane of the trace data into 8 zones and determine the maximum signal amplitude in each.
2303
Identify the two largest amplitude zones.
2304
If the two zones are adjacent set the Begin at the left-most zone maximum, and End to the last sample.
2305
If the two zones are not adjacent set Begin at the left most zone maximum and End at the right most zone maximum.
2306
Examine 400 samples right of Begin for a 2nd primer peak (amplitude > 30% of 1st primer peak). Move Begin to 2nd peak if one exists.
2307
Move Begin right to the 1st sample which is below the mean of all samples between Begin and (Begin+End)/2.
2308
Move End left 350 samples. The tail end of the data is, at this juncture, is not worth reading.

Figure 23

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/16933

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :C12Q 1/68; G06F 17/17
US CL : 435/6; 364/400
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 435/6; 364/400

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS, MEDLINE, BIOSIS
SEARCH TERMS: SEQUENCE, FUZZY LOGIC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,365,455 A (TIBBETTS ET AL.) 15 November 1994, columns 6-22.	1-20
A	US 5,273,632 A (STOCKHAM ET AL.) 28 December 1993, columns 3-16.	1-20

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 16 DECEMBER 1997	Date of mailing of the international search report 21 JAN 1998
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JOHN S. BRUSCA Telephone No. (703) 308-0196

Form PCT/ISA/210 (second sheet)(July 1992)*